UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

GESTIÓN DE LA INFORMACIÓN EN APLICACIONES WEB: ETIQUETADO JERÁRQUICO

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN: TELEMÁTICA

Autor: Jorge Fonseca Mesonero
Tutor: Carlos Alario Hoyos

Septiembre de 2015

Agradecimientos

A mi familia por la educación y los valores recibidos que me han formado como persona.

A mis compañeros de universidad por todos los momentos compartidos y hacer de la carrera una experiencia inolvidable.

A mis tutores por ayudarme en este último tramo de carrera y las enseñanzas adquiridas.

Y en general a todos los que han compartido estos años de mi vida; profesores, amigos y familia. No hace falta que os nombre a ninguno porque sabéis quienes sois y no terminaría nunca de escribir nombres. Gracias a todos.

Resumen

En la actualidad vivimos rodeados de un gran volumen de información que no para de crecer. Cada día nos enfrentamos a la necesidad de organizar y procesar mejor la información que recogemos de Internet para poder extraer datos valiosos para nosotros. Es por ello, que en los últimos años se ha estado trabajando mucho en desarrollar herramientas que sean capaces de procesar grandes volúmenes de datos y de devolver información más precisa y concreta a nuestras búsquedas.

En este proyecto se plantea el diseño y desarrollo de una extensión para un navegador web que nos permita hacer una clasificación jerárquica de la información de distintos sitios y aplicaciones web donde exista previamente información clasificada mediante etiquetas. Esta extensión tiene como finalidad facilitar la búsqueda de información fusionando los sistemas de clasificación jerárquica y por etiquetado.

Abstract

Today we are surrounded by a large volume of information that it is growing. Every day we are faced with the need to organize and process information we collect from the Internet to extract valuable information for us. That is why in recent years has been working hard to develop tools that are capable of processing large volumes of data and return more precise and concrete information to our searches.

This project proposes the design and development of a web browser extension that allows us to make a ranking of information from different sites and web applications where previously exists classified information through labels. This extension is intended to facilitate the search for information by merging the hierarchical and labeling classification systems.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1. Contexto

En la actualidad vivimos rodeados de un gran volumen de información que no para de crecer cada día. Internet es el máximo exponente de esta situación, donde en tan sólo dos días se puede generar tanta información como la generada por toda la Humanidad hasta el año 2003 [1]. En tan sólo un minuto se pueden generar más de 200 millones de correos o más de 500.000 imágenes tal y como se puede apreciar en la figura 1.1.

Todos estos datos serían inútiles e inservibles si no fueran procesados para sacar información de ellos y hacerlos accesibles, pero tal volumen de datos hace que las bases de datos tradicionales y el procesamiento centralizado de los datos sean técnicas insuficientes para este fenómeno, originando el concepto de *Big Data*. Este concepto engloba a la gestión y análisis de enormes volúmenes de datos, los cuales no pueden ser tratados de manera convencional con herramientas tradicionales de gestión de base de datos [2]. Por tanto, *Big Data* no sólo hace referencia a grandes volúmenes de datos en sí, sino que también se refiere a los procesos relacionados para tratar dicha información. Las características de este término son conocidas como las 5 V´s [3], volumen, variedad, velocidad, veracidad y valor.

A este fenómeno se suma la tendencia hacia la Web semántica [4], donde se pretende que las aplicaciones sean capaces de procesar la información y devolver la información que se precisa, en lugar de responder con toda la posible información relacionada. Con esta tendencia se pretende conseguir una comunicación más útil e intuitiva con los programas informáticos.

Es por todo ello que en los últimos años se ha estado trabajando mucho en desarrollar herramientas que traten de sacar información útil a grandes cantidades de datos, ya que si los datos no son debidamente procesados no se podría extraer ninguna información valiosa de ellos.

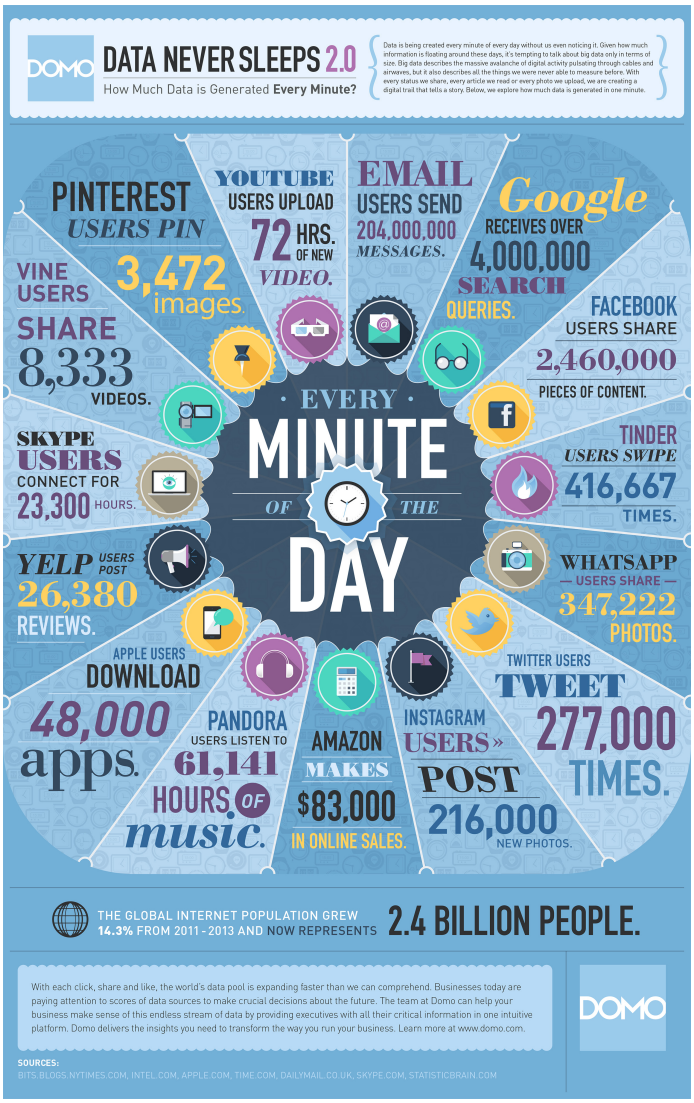

Figura 1.1: Infografía de datos generados cada minuto en 2014 [DOMO]

1.2. Motivación

Como hemos visto, Internet es el principal foco donde se pueden concentrar grandes volúmenes de datos e información. Es mediante aplicaciones en red o simplemente páginas/aplicaciones web como se enlaza todo este contenido generado en Internet. Para que toda esta información pueda ser recuperada es necesario que se utilicen sistemas de clasificación de la información, ya que de lo contrario no sería posible recuperar la información fácilmente.

Los principales modelos de clasificación de la información que podemos encontrar en las aplicaciones web son el etiquetado y la clasificación jerárquica [5]. Con estos modelos un usuario puede encontrar el contenido de una búsqueda concreta sin demasiadas dificultades, pero en el caso de realizar una búsqueda más ambigua o explorativa podría suponer un esfuerzo mayor para el usuario. Estos modelos de clasificación generalmente vienen dados por defecto en las aplicaciones web sin que el usuario pueda escoger qué modelo le interesa más, o incluso sin poder personalizar el sistema de clasificación, no permitiendo limitar así los inconvenientes que puedan llegar a tener un modelo u otro.

Es por esto que se ha querido realizar una herramienta que ayude a un usuario a sacar valor de una aplicación web permitiéndole fusionar los dos sistemas de clasificación más utilizados (etiquetado y clasificación jerárquica) para intentar facilitar sus búsquedas. Dado que el contenido web es accedido mediante navegadores, se ha decidido que la herramienta desarrollada consista en una extensión para el navegador, concretamente el popular navegador *Mozilla Firefox*, de forma que se pueda interactuar con la aplicación web externamente sin necesidad de incluir la herramienta en las propias aplicaciones.

De las posibles aplicaciones web que existen, se ha decidido aplicar esta herramienta concretamente al portal *Delicious*. Se podría haber escogido cualquier otra aplicación que tuviera como sistema de clasificación el etiquetado, pero se eligió ésta por su similitud a la problemática comentada en el contexto, ya que consiste en un almacén de contenido web donde se pueden almacenar una gran cantidad de enlaces, los cuales si no son debidamente tratados mediante un sistema de clasificación no servirían y no aportarían nada al usuario.

1.3. Objetivos

El principal objetivo de este proyecto es desarrollar una extensión para un navegador, que nos permita representar un modelo de clasificación de la información jerárquico en páginas web donde haya información previamente clasificada mediante etiquetas. En concreto, se pretende implementar una extensión para un navegador que actúe sobre una aplicación web, permitiendo al usuario relacionar de forma jerárquica las diferentes etiquetas que tuviera dicha aplicación para un mejor acceso a la información.

Dentro de los objetivos secundarios de este proyecto está el realizar un análisis profundo de las clasificaciones de información, tanto por etiquetado como jerárquicas, valorando las posibles ventajas y desventajas de ambas. Dentro de este estudio se analizará la posibilidad de fusionar ambos conceptos y ver qué podría aportar dicha fusión. Otro objetivo secundario será realizar un análisis de las tecnologías a utilizar para la implementación de la extensión.

Como objetivo secundario está también la realización de un diseño modular de la extensión para permitir que con pequeños módulos añadidos pueda ser compatible para diferentes aplicaciones web. De esta forma conseguiremos un valor añadido en la extensión ya que podría ser fácilmente adaptable a otras aplicaciones web sin necesidad de rehacer toda la lógica. Este diseño de la ex-

tensión basado en la arquitectura "Modelo Vista Controlador" [6] nos permitirá adaptar la extensión a los posibles cambios de versiones que nos encontremos en la aplicación web realizando pequeñas modificaciones.

1.4. Estructura de la memoria

A continuación realizamos un breve resumen de los sucesivos capítulos de la presente memoria para facilitar su lectura.

- En el segundo capítulo se realizará un análisis teórico del proyecto. En este capítulo se analizarán los fundamentos en los que está basado el proyecto, explicando cómo llegamos a la solución implementada y describiendo los conceptos en los cuales nos basamos: la clasificación por etiquetado y la clasificación jerárquica. Por último, definiremos la clasificación por etiquetado jerárquico, la cual utilizaremos en la solución a implementar.

- Seguidamente, en el tercer capítulo hablaremos de las tecnologías implicadas en el proyecto. En este capítulo se describirán y analizarán las diferentes tecnologías utilizadas para la realización de la extensión. Se hablará del navegador *Mozilla Firefox* y de cómo crear extensiones para dicho navegador. Describiremos los lenguajes utilizados para la implementación de la extensión. También hablaremos de la tecnología empleada para el almacenamiento de la información.

- Continuaremos con el cuarto capítulo donde se indicarán los requisitos que deberá cumplir la extensión y donde se realizará un diseño de la extensión para explicar cómo funciona y cómo interactúan las diferentes partes de la misma.

- Después, en el quinto capítulo describiremos con detalle la implementación seguida para desarrollar la extensión. Definiremos la estructura de la extensión incluyendo una explicación de los diferentes ficheros utilizados y concluiremos con unas consideraciones generales en las que contaremos algunos problemas que nos encontramos en el desarrollo de la extensión.

- A continuación, en el sexto capítulo se recogerán las conclusiones a las que se ha llegado a la finalización del proyecto, así como las posibles líneas futuras que se pueden desarrollar en el mismo para su mejora o ampliación de funcionalidades.

- En el último y séptimo capítulo detallaremos los medios empleados, describiremos las fases de desarrollo llevadas a cabo en el proyecto y calcularemos el presupuesto final desglosando los diferentes costes.

- Tras finalizar los capítulos podremos encontrar un glosario de términos para aclarar las siglas utilizadas durante la memoria y una bibliografía con un listado de referencias consultadas para el desarrollo del proyecto.

- Finalmente incluimos un anexo con los manuales de la extensión implementada. En este anexo se incluirá un manual de instalación para indicar a un usuario cómo conseguir instalar la extensión en el navegador *Mozilla Firefox*, y un manual de usuario para detallar el funcionamiento básico de la extensión desarrollada. Dichos manuales incluirán imágenes aclaratorias acompañando a las explicaciones.

Capítulo 2

Análisis teórico del proyecto

En este capítulo analizaremos el fundamento en el cual está basado el proyecto, explicando cómo llegamos a la solución implementada y describiendo los conceptos en los cuales nos basamos. Empezaremos con una introducción temática explicando la motivación del proyecto. A continuación comentaremos en qué se basan la clasificación por etiquetado y la clasificación jerárquica, sus ámbitos de aplicación y sus ventajas y desventajas. Una vez explicados estos conceptos, explicaremos el nuevo sistema de clasificación híbrido que proponemos: la clasificación por etiquetado jerárquico. Y finalmente terminaremos con unas breves conclusiones sobre la clasificación de la información y el nuevo método de clasificación híbrido.

2.1. Introducción

En la actualidad vivimos en la sociedad de la información [7], donde la información y el conocimiento tienen un lugar privilegiado en la sociedad y en la cultura. Esta sociedad es la sucesora de la sociedad industrial y sus principales características son:

- La información es su materia prima, es decir, las tecnologías actúan sobre la información y no al revés.
- La información es algo esencial en la vida humana.

Es por ello que en la actualidad la información es tan importante y existen muchos procesos para tratar dicha información para que pueda llegar a ser útil y fácilmente accesible. Como ejemplos tendríamos la minería de datos [8], la cual es una ciencia que tiene como fin extraer información útil de un gran volumen de datos; y la clasificación de la información [9], gracias a la cual se puede caracterizar la información según una temática para poder identificarla y recuperarla en el futuro y así extraer valor de ella. Y es en la clasificación de la información donde nos centraremos en este proyecto para intentar facilitar el acceso a la misma.

Existen varios tipos de clasificación de la información, pero en entornos informáticos son la clasificación por etiquetado y la clasificación jerárquica los más extendidos. Y es a partir de ellos como llegaremos al nuevo modelo de clasificación híbrido que se propone como parte de este trabajo. Estos tipos de clasificación se explican a continuación.

2.2. Clasificación por Etiquetado

Antes de hablar de la clasificación por etiquetado [5], introduciremos el concepto de etiqueta. Una etiqueta o *tag*, es una palabra clave asignada a un dato almacenado con el fin de facilitar su recuperación. Estas etiquetas son elegidas de manera informal y personal por los propios usuarios. La acción de etiquetar datos se asocia a los sitios Web 2.0 al ser webs sociales y colaborativas donde se permite agregar y cambiar información.

Una vez explicado el concepto de etiqueta, es fácil comprender cómo funciona el sistema de clasificación por etiquetado. Esta clasificación consiste en asignar a todo el contenido almacenado una o varias etiquetas descriptoras del mismo, para así poder obtener en relación a un concepto definido como etiqueta toda la información relacionada. En estos sistemas de clasificación existen dos formas de ir obteniendo resultados según las etiquetas: uno es mediante un listado de todas las etiquetas existentes en el que pinchando en las diferentes etiquetas se van mostrando los resultados de todos los contenidos relacionados con ellas; y el otro es mediante un buscador de etiquetas donde se escribe un concepto y se muestra todo el contenido relacionado con él, en caso de que existiera tal concepto definido como etiqueta. En la figura 2.1, se puede ver un ejemplo de un listado de etiquetas en forma de nube, con las etiquetas más utilizadas resaltadas sobre las demás.

Figura 2.1: Ejemplo de una colección de etiquetas resaltando las más usadas [Wikipedia]

En los sitios web que permiten etiquetar sus datos, la colección de etiquetas se llama folcsonomía [10]. La folcsonomía se caracteriza porque los usuarios comparten las categorizaciones, a diferencia de otros sistemas de categorización como puede ser *Gmail*, donde las etiquetas son privadas y no compartidas. Existen dos tipos de folcsonomía.

- **Folcsonomía amplia:** son los propios usuarios los que etiquetan el contenido sin la influencia del creador. Su ejemplo más identificativo sería *del.icio.us*, un sitio para compartir enlaces favoritos.
- **Folcsonomía estrecha:** el contenido es etiquetado sólo por el creador o un número reducido de personas. Un ejemplo de ello sería *Flickr*, un sitio para compartir fotos que son etiquetadas por el autor.

Las ventajas de la clasificación por etiquetado son:

- No es necesario que exista un esquema de clasificación previo. Se asignan las etiquetas necesarias para cada dato concreto según sea necesario.
- Permite reclasificar un dato con solo modificar sus etiquetas. Todas las conexiones entre el dato reclasificado y otros datos almacenados se actualizan automáticamente sin necesidad de intervención de la persona que ha realizado la tarea. Tampoco es necesario cambiar el dato de categoría dentro de una compleja jerarquía de categorías.
- Resulta más flexible y sencillo en comparación con otros sistemas de clasificación.
- Incluye todas las etiquetas propuestas sin dejar ninguna fuera.
- La información clasificada puede crecer todo lo que sea necesario sin que el sistema pierda calidad y se vuelva complejo el acceso a la misma.

Por contra, las desventajas de este sistema de clasificación son:

- La falta de control terminológico ya que las etiquetas no tienen un significado semántico, único y explícito. Esto disminuye la eficiencia de la búsqueda provocando la aparición de:
  - Sinonimias, múltiples etiquetas del mismo concepto.
  - Homonimia, misma etiqueta con diferente significado (origen etimológico distinto).
  - Polisemia, misma etiqueta con múltiples significados relacionados (mismo origen etimológico).
- La heterogeneidad de usuarios y contextos produce una excesiva etiquetación, ya que pueden aparecer un gran número de etiquetas que solo tenga sentido para unos pocos.

2.3. Clasificación Jerárquica

La clasificación jerárquica [5] consiste en separar un conjunto de datos en clases y subclases siguiendo unas variables determinadas por los criterios de clasificación. Su nombre es debido a que se obtienen diferentes niveles o subclases,

dando lugar a un grupo de objetos ordenados por rangos donde todos están subordinados al objeto superior, a excepción del primero.

Con este sistema de clasificación toda la información está representada en un árbol jerárquico, siendo dividida en una serie de conceptos los cuales serían los nodos del árbol. La información deseada se obtiene recorriendo el árbol a través de los diferentes niveles. En los niveles más altos estarían las temáticas más amplias y según se va profundizando en el árbol se llega a temas más específicos. En la figura 2.2, se puede ver un ejemplo gráfico de la estructura jerárquica de este sistema de clasificación, partiendo de "*abcdef*" como el nivel más alto o superior.

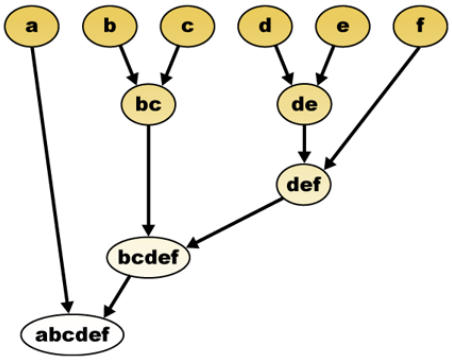

Figura 2.2: Ejemplo gráfico de la estructura jerárquica [Wikipedia]

Este sistema de clasificación lo podemos encontrar en todos los Sistemas Operativos, ya que es el usado para recorrer los directorios. También se puede encontrar en aplicaciones informáticas para recorrer las diferentes herramientas o funcionalidades que tenga.

Las ventajas de la clasificación jerárquica son:

- Permite una búsqueda explorativa. Esto es idóneo para cuando el usuario no sabe exactamente lo que busca o no es capaz de expresarlo con palabras. De esta manera el usuario podrá ir explorando visualmente las opciones pudiendo intuir dónde encontrar el contenido de su interés.
- Se puede acceder fácilmente al contenido relacionado con nuestra búsqueda sin tener que realizar otra, solamente habría que subir un nivel en el árbol.

Por contra, las desventajas de este sistema de clasificación son:

- Tiene un elevado coste inicial y de mantenimiento debido al requerimiento de definir la estructura al comienzo y a posibles actualizaciones en la estructura.
- Se deben escoger los criterios de clasificación más apropiados descartando otros, haciendo que dichos criterios puedan no adaptarse bien a todos los usuarios si es un grupo heterogéneo.

- Es poco escalable ya que si la información clasificada creciera mucho y requiriera crear nuevos criterios de clasificación la búsqueda se podría volver compleja y costosa.

2.4. Clasificación por etiquetado jerárquico

Como hemos visto, la clasificación por etiquetado y la clasificación jerárquica tienen desventajas que casi se contrarrestan entre ellas si se complementaran ambos sistemas de clasificación. Es por ello que implementaremos lo que nosotros llamaremos la clasificación por etiquetado jerárquico, la cual permite crear un sistema de clasificación bastante potente con muy pocas desventajas.

En la actualidad podemos encontrar los primeros ejemplos de este sistema en la taxonomía corporativa debido a que en las empresas se están dando cuenta que el exceso de información combinado con el analfabetismo informacional dan como resultado una baja productividad [11]. Y es por ello que surge la taxonomía corporativa como una metodología para el almacenamiento y clasificación de los proyectos, la cual es muy similar a la clasificación por etiquetado jerárquico. Esta metodología consiste en catalogar la información en base a un trinomio básico compuesto por el contexto, la audiencia y los contenidos. En base al trinomio comentado se identifican los criterios de clasificación, se realiza una extracción y control del léxico a utilizar para la catalogación de la información y se implementa el desarrollo de la estructura de la taxonomía fusionando la relación jerárquica y el etiquetado asociativo. En la figura 2.3, se puede observar un ejemplo de la aplicación de esta metodología.

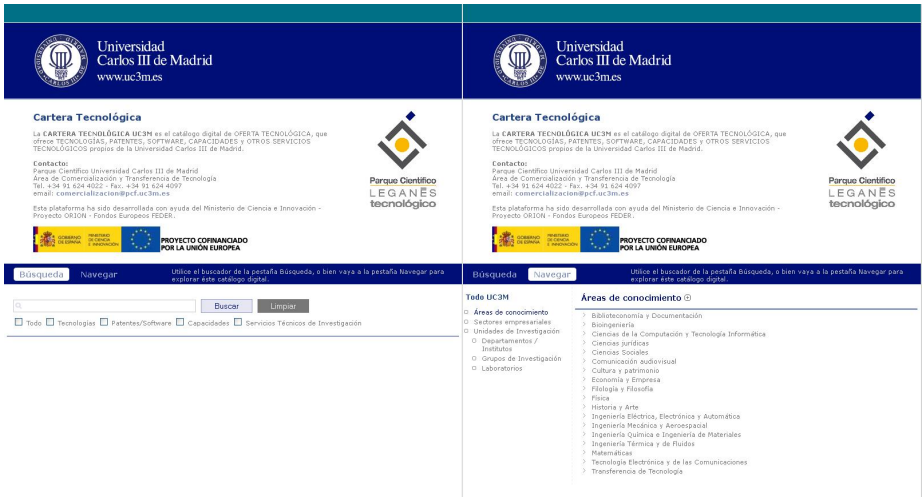

Figura 2.3: Ejemplo de una aplicación empleando la taxonomía corporativa como metodología [Proyecto ORIÓN-UC3M]

El concepto de la clasificación por etiquetado jerárquico consiste en rela-

cionar todo el contenido, clasificado previamente por etiquetas personales del usuario, de forma jerárquica para facilitar sus búsquedas. De esta forma, al potente sistema de etiquetado en el que toda la información está catalogada para su fácil recuperación, se le añade la ventaja de la búsqueda explorativa que ofrece el sistema jerárquico. No sólo se podrá obtener la información que se busca utilizando los términos necesarios, sino que también se podrá obtener en el caso de no saber describir lo que se busca o querer obtener contenido relacionado. Además se disminuye el efecto de la falta de control terminológico del sistema por etiquetado, ya que al estar el contenido relacionado se podría deducir el significado de la etiqueta cuando ésta tenga varios significados.

Un posible ejemplo sería en el que alguien quisiera recuperar todas las fotos de su viaje por Europa, pero no ha catalogado todas esas fotos con la etiqueta "*Europa*", sino que ha empleado también etiquetas como "*Francia*" o "*Inglaterra*". Si hiciera una búsqueda por "*Europa*" no le saldrían todas las fotos, y tendría que ir haciendo más búsquedas. Con nuestro sistema, tendría la opción de seguir la hipotética ruta de "*Viajes/Fotos/Europa*" para obtener todas las fotos, ya que dentro de Europa se encontrarían las categorías Francia e Inglaterra.

A pesar de tener las ventajas mencionadas, este sistema híbrido posee algunas desventajas fruto de una incompatibilidad puntual entre ambos sistemas. Las desventajas de la clasificación por etiquetado jerárquico vienen heredadas de los sistemas de clasificación en los que se basa, como son la necesidad de tener un control terminológico para evitar duplicidades y el coste inicial debido a la conveniencia de definir una estructura básica previamente. El control terminológico se hace necesario ya que al relacionar los términos de clasificación es preciso que no existan ambigüedades y aparezca contenido referente a una etiqueta pero sin relación con la etiqueta superior. Por este inconveniente es necesario definir una estructura previa, suponiendo un coste inicial a tener en cuenta. Para explicar estas desventajas utilizaremos un ejemplo que nos ayudará a comprenderlo mejor.

En un sistema jerárquico un usuario puede tener un directorio sobre exámenes de la carrera y dentro las diferentes asignaturas, que a su vez éstas contienen los exámenes separados por cursos. Es decir, puede tener un directorio de la siguiente forma:

Exámenes

- Álgebra
  - Curso 2011
  - Curso 2012
- Electrónica
  - Curso 2011
  - Curso 2012

Si este ejemplo jerárquico lo lleváramos al sistema por etiquetado no habría ningún problema ya que los recursos estarían bien etiquetados y se podrían

recuperar con una búsqueda compuesta por dos términos, pero al llevarlo al sistema híbrido surge una incompatibilidad debido a que por definición en el sistema híbrido no puede aparecer una etiqueta por duplicado en la estructura jerárquica. Esto es debido a que en la estructura puramente jerárquica, aunque haya un concepto duplicado, éste contendrá información diferente salvo que sea una duplicidad. En el ejemplo arriba mencionado, a pesar de aparecer los directorios "*Curso 2011*" y "*Curso 2012*" por duplicado, éstos contienen exámenes de diferentes asignaturas. Pero si este ejemplo lo llevamos al sistema híbrido, las etiquetas "*Curso 2011*" y "*Curso 2012*" tendrán los exámenes de las asignaturas "*Álgebra*" y "*Electrónica*" simultáneamente. Con lo cual, si recorrieras el árbol jerárquico y seleccionaras el directorio "*Exámenes/Álgebra/Curso 2011*" aparecerían también los exámenes de "*Electrónica*" que no son los deseados. Y es por este motivo por el cual no puede aparecer ninguna duplicidad en la estructura jerárquica del sistema híbrido. Para solventar esta incompatibilidad, bastaría con redefinir las etiquetas ambiguas de una forma más precisa para que no exista contenido demasiado heterogéneo en una misma etiqueta.

2.5. Conclusiones

El uso que le damos a Internet [4, 12] ha ido evolucionando hasta convertirse en una necesidad para la mayoría de las personas. En el comienzo, la Web 1.0 era únicamente de lectura donde se podía acceder a la información pero los usuarios no podían interactuar. Ésta evolucionó a la Web 2.0, que es la que usamos actualmente, donde los usuarios pueden interactuar y agregar o modificar información convirtiéndose en una web social y colaborativa. Y ahora debido al auge de la información que vivimos, la siguiente evolución es la Web 3.0 la cual combina los conceptos de contenido semántico, inteligencia artificial, inteligencia colectiva y gestión del conocimiento. Esta evolución se conoce también como Web semántica, ya que cuando realicemos una búsqueda interpretará las palabras que usemos para crear y mostrar contenido relevante en lugar de ofrecernos simplemente una gran cantidad de contenido relacionado con las palabras utilizadas sin interpretar el significado de la búsqueda. Las siguientes evoluciones futuras serán la Web 4.0 y Web 5.0 [13], siendo la primera una Web orientada a la comunicación entre inteligencias artificiales y personas en cualquier lugar y momento y la segunda una Web capaz de incorporar emociones para interactuar con el usuario.

Dado que seguimos todavía en la Web 2.0[1], donde el acceso a la información se obtiene de forma similar al etiquetado mostrando resultados que contengan las palabras utilizadas en una búsqueda, nos surgió la idea de complementar ese acceso a la información con el sistema de clasificación por etiquetado jerárquico.

Dicho todo esto, podemos concluir que a pesar de haber ido surgiendo diferentes métodos de clasificación de la información de manera independiente y

[1] Todavía no hemos dado el salto definitivo a la Web 3.0 aunque cada vez está más implantada.

con aplicaciones diferentes, el futuro de la generación del conocimiento consistirá en relacionar todos estos sistemas para brindar un mecanismo potente capaz de ofrecer la información requerida. La construcción de este sistema requiere un esfuerzo inicial y de mantenimiento para la elección del léxico adecuado y el establecimiento de las relaciones entre los diferentes conceptos, pero a cambio se obtiene un rendimiento mejor en la obtención de la información ya que serán más precisos los resultados.

Capítulo 3

Tecnologías implicadas

En este capítulo se describirán y analizarán las distintas tecnologías utilizadas para la realización de la extensión. Comentaremos las características del navegador *Mozilla Firefox* y algunos conceptos importantes utilizados, como es el *Drag & Drop*, así como la estructura básica que debe tener una extensión realizada para dicho navegador. También detallaremos aspectos del lenguaje de programación utilizado, *JavaScript*, de los lenguajes de consulta utilizados para encontrar información en documentos XML y en bases de datos, XPath y SQL respectivamente, y del lenguaje de marcas utilizado para la interfaz de usuario, XUL. Seguidamente hablaremos de la tecnología utilizada para el almacenamiento de los datos, SQLite. Finalmente haremos un resumen de las tecnologías implicadas para dar una visión global destacando los aspectos más importantes y comentando las posibles herramientas a utilizar para facilitar su desarrollo.

3.1. El navegador Mozilla Firefox

3.1.1. Elección del navegador

Para la realización de la extensión teníamos varios navegadores posibles, tales como *Mozilla Firefox*, *Google Chrome*, *Internet Explorer*, *Opera* y *Safari* entre otros. Para la elección de un único navegador aplicamos una serie de criterios personales como son la cantidad de usuarios que usan el navegador, el perfil de dicho usuario y la capacidad de personalización del navegador. En la figura 3.1, se pueden observar unas gráficas sobre las estadísticas globales de utilización de los navegadores para poder aplicar el primer criterio de selección[1].

Es por ello que, tras aplicar el primer criterio de selección, redujimos las opciones de navegadores a solamente tres: *Internet Explorer*, *Mozilla Firefox* y *Google Chrome*, siendo el resto de navegadores, como *Opera* y *Safari*, descarta-

[1] Firefox, Internet Explorer y Chrome siguen siendo los tres navegadores más usados en la actualidad, aunque en comparación con el inicio del proyecto Chrome ha desbancado a Internet Explorer mientras que Firefox se ha mantenido

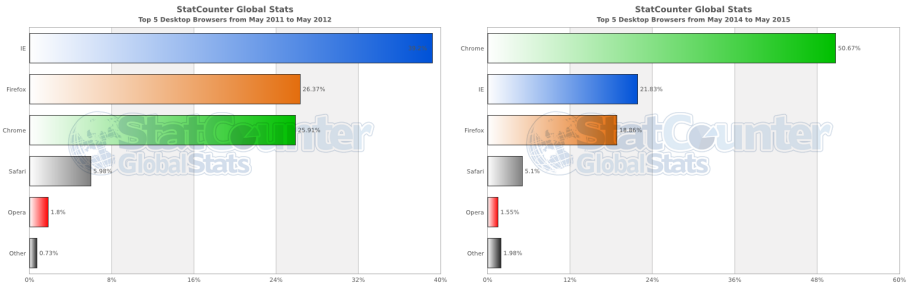

Figura 3.1: Estadísticas globales sobre utilización de navegadores [StatCounter]

dos debido a que cuentan con muy pocos usuarios y queríamos llegar al mayor número de usuarios posibles.

El siguiente paso fue descartar el navegador *Internet Explorer* debido al perfil de usuario que posee dicho navegador. La mayoría de sus usuarios lo utilizan porque es el que viene por defecto en sus equipos[2] y personalmente consideramos que dichos usuarios no están muy familiarizados con la personalización de los navegadores. A esto hay que añadir los problemas de incompatibilidad que tiene el navegador con muchas páginas web.

Llegados a este punto, contábamos con dos posibles navegadores, *Mozilla Firefox* o *Google Chrome*. Ambos navegadores actualmente son muy similares en cuanto a prestaciones y la elección de un navegador u otro suele estar básicamente fundamentada en los gustos del usuario. La elección de *Mozilla Firefox* fue hecha desde el principio, entre otros motivos porque en el comienzo del proyecto *Google Chrome* no era tan personalizable con la inclusión de complementos. Además, según pudimos comprobar, el navegador *Firefox* cuenta con una extensa documentación para los desarrolladores, incluyendo pequeños ejemplos de las distintas funcionalidades, e incluso parte de la documentación cuenta con traducciones en otros idiomas, mientras que la documentación de *Chrome* es bastante más limitada.

3.1.2. Propiedades de Mozilla Firefox

A continuación hablaremos de las propiedades del navegador escogido para la realización de la extensión.

Mozilla Firefox es un navegador web libre y de código abierto descendiente de *Mozilla Application Suite*[3] y desarrollado por la Fundación *Mozilla*[4]. Cualquier persona o compañía puede colaborar en el proyecto *Mozilla*, ya sea aportando

[2] Alrededor del 90 % de la población usa Windows, según datos consultados en StatCounter

[3] Mozilla Application Suite es un navegador web y una plataforma de desarrollo libre y de código abierto. Por decisión de la Fundación Mozilla esta suite ha dejado de ser desarrollada siendo actualmente su sucesor SeaMonkey.

[4] Mozilla Foundation es una organización sin ánimo de lucro dedicada a la creación de software libre. Tiene como misión "mantener la elección y la innovación en Internet".

código, probando los productos, escribiendo documentación o de cualquier otra manera. El trabajo desinteresado de los contribuidores de *Mozilla* hace que sea una organización sin fines de lucro. Las características del software que cumplen los productos de *Mozilla* son:

- el código abierto;
- el respeto por los estándares y la portabilidad;
- la posibilidad para la interacción del software en múltiples plataformas.

Según la página oficial de *Mozilla Firefox* [14], las principales características del navegador son:

- Una navegación más fácil gracias una interfaz intuitiva, navegación en pestañas, sincronización entre diferentes dispositivos, búsquedas inteligentes, lector de canales RSS, gestor de descargas, corrector ortográfico y posibilidad de restaurar una sesión.
- Alto rendimiento permitiendo tiempos de inicio más rápidos, mejoras en la velocidad de carga de las páginas, aceleración de hardware para las aceleraciones gráficas y protección frente a fallos.
- Seguridad avanzada mediante controles parentales, integración con antivirus, protección frente a suplantación de identidad (*antiphising*), conexiones seguras a sitios web, navegación privada, limpieza del historial reciente, actualizaciones automáticas, detecciones de *plugin* obsoletos, bloqueador de ventanas emergentes y un gestor de contraseñas.
- Amplia personalización mediante un administrador de complementos, utilización de temas e interfaz modificable. Cuenta con gran variedad de temas y complementos para la personalización y aumento de la funcionalidad.
- A la vanguardia tecnológica como por ejemplo la posibilidad de interpretar HTML5.
- El acceso universal que permite una gran variedad de tipografías.
- La asistencia y ayuda al usuario, contando incluso con un chat en directo en el que los miembros de la comunidad resuelven dudas.

A pesar de todas estas características, lo que más se le ha criticado al navegador ha sido su velocidad de carga y su elevado consumo de recursos. Desventajas que le hizo perder usuarios (se mudaron a *Google Chrome* por ser más ligero y rápido[5]) a pesar de haber realizado bastantes mejoras en estos dos aspectos con las sucesivas versiones.

[5]En la actualidad Google Chrome consume mucha mas memoria que Mozilla Firefox pero en el inicio del proyecto en el año 2012 era al revés

El código fuente de *Firefox* está disponible libremente bajo la triple licencia de *Mozilla* como un programa libre y de código abierto. La versión activa en el momento de escribir esta memoria es la 41.0 publicada el 22-09-2015, sin embargo en el contexto de este proyecto empezamos usando la versión 3.6 que era la existente en el momento del desarrollo del mismo aunque fuimos adaptando la extensión para las nuevas versiones que iban saliendo[6]. El cambio tan grande de versión se debe a que *Firefox* cambió su política de actualizaciones por la fuerte competencia con *Google Chrome*; pasaron de sacar actualizaciones cada mucho tiempo sin dar grandes saltos en la numeración a publicarlas cada pocas semanas con versiones que cambiaban en enteros.

3.1.3. Desarrollo de una extensión para el navegador Mozilla Firefox

Para la realización de una extensión en *Firefox*, hay que seguir una estructura básica [15] y desarrollarla según las necesidades de nuestra extensión. Las extensiones de *Firefox* se empaquetan y distribuyen en archivos ZIP, con extensión xpi. La estructura que siguen es:

extension.xpi:

/install.rdf
/chrome.manifest
/components/*
/defaults/preferences/*.js
/plugins/*
/chrome/
/chrome/content/
/chrome/skin/
/chrome/locale/

La finalidad de cada fichero/directorio de la estructura es:

- Los ficheros de configuración *install.rdf* y *chrome.manifest*, el primero describe los aspectos de instalación y el segundo registra el contenido creado en la extensión. Ambos los detallaremos a continuación por ser de gran importancia.
- El directorio opcional *components*, que contendría las interfaces XPCOM [16] si hubiera acciones que no pudiéramos implementar con *JavaScript* como por ejemplo crear una aplicación de correo en la que haya que conectarse con los servidores.
- El directorio *defaults/preferences*, que contendrá los archivos predeterminados, con extensión *.js, que se utilizan como semillas de un perfil de

[6] Comenzamos a trabajar en este proyecto en el segundo trimestre del año 2010

usuario. Un ejemplo de línea de código seria: `pref("extensions.sample.username", "Joe"); //Un string de preferencia`

- El directorio opcional *plugins*, que contendría los plugins que se necesitasen en la extensión para ejecutar alguna aplicación.
- El directorio *chrome*[7], para indicar al navegador *Firefox* que lo que se encuentra dentro deberá ser interpretado por él y gestionado de forma especial.
- El directorio *chrome/content*, contendrá todo el código de definición de la interfaz de usuario (archivos *.xul) y toda la lógica necesaria para implementar el comportamiento de la extensión (archivos *.js)
- El directorio *chrome/skin*, contendrá los estilos utilizados en la interfaz (archivos *.css) y las imágenes utilizadas.
- El directorio *chrome/locale*, contendrá los archivos de configuración de idiomas para permitir generar extensiones multilenguaje.

El archivo install.rdf

El archivo *install.rdf* contiene información sobre la extensión a instalar y debe estar en la carpeta raíz. Define los metadatos que identifican el complemento, las versiones compatibles, las actualizaciones, la plataforma y los desarrolladores entre otras.

Un ejemplo del contenido sería:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-
       ns#" xmlns:em="http://www.mozilla.org/2004/em-
       rdf#">
3    <Description about="urn:mozilla:install-manifest">
4      <em:id>miextension@undominio</em:id>
5      <em:type>2</em:type>
6      <em:name>Nombre de la Extensión</em:name>
7      <em:version>1.0</em:version>
8      <em:creator>Nombre del autor</em:creator>
9      <em:description>Descripción de la extensión.</
           em:description>
10     <em:targetApplication>
11       <Description>
12         <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384
               }</em:id> <!-- Firefox -->
13         <em:minVersion>Versión mínima soportada</
               em:minVersion>
```

[7] A pesar de tener el mismo nombre que el navegador de Google este directorio no tiene nada que ver con él. Explicaremos su funcionalidad más adelante.

```
14          <em:maxVersion>Versión máxima soportada</
                em:maxVersion>
15        </Description>
16      </em:targetApplication>
17    </Description>
18  </RDF>
```

A continuación detallamos las propiedades más relevantes.

- **id:** el identificador de la extensión que debe ser único, será un GUID o un identificador con formato de correo electrónico.
- **type:** un valor de tipo entero que representa el tipo de complemento. Los posibles valores son: extensiones el 2, temas el 4, configuración de idiomas el 8, plugins el 16, paquetes de múltiples elementos el 32 y corrector ortográfico el 64.
- **name:** el nombre del complemento.
- **version:** indica la versión de la extensión.
- **creator:** el nombre del creador de la extensión.
- **description:** una breve descripción del complemento.
- **targetApplication:** especifica una aplicación a la cual se dirige el complemento. Las propiedades *minVersion* y *maxVersion* especifican el rango de versiones de la aplicación con las que funciona correctamente la extensión.

El archivo chrome.manifest

Antes de comentar y definir el archivo *chrome.manifest*, explicaremos qué son las chrome URIs ya que son los identificadores de recursos que se utilizan en el fichero. Los archivos XUL forman parte de Chrome Packages, paquetes de componentes de interfaz del usuario, los cuales se cargan a través de la dirección *chrome:// URIs*. En lugar de cargar el navegador y las extensiones desde el disco utilizando un archivo (*file://*), ya que la ubicación de *Firefox* puede cambiar de una plataforma a otra y de un sistema a otro, los desarrolladores se decantaron por cargar los componentes basándose en un identificador (URI) creado respecto al contexto XUL, que la aplicación ya conoce. Así por ejemplo, la ventana del navegador es: *chrome://browser/content/browser.xul*.

Las URIs chrome constan de varias partes:

- En primer lugar, el esquema URI (*chrome*) informa a la librería de red de *Firefox* de que es un 'Chrome URI' y que el contenido que se cargue debe ser manejado de manera especial.

- En segundo lugar, un nombre de paquete (en el ejemplo del navegador sería *browser*), que identifica al componente en la interfaz del usuario. Este nombre debe ser único; de esa manera se evitará el conflicto entre componentes (navegador y/o extensiones).
- En tercer lugar, el tipo de información que ofrece el archivo. Hay tres tipos:
  - *content*, archivos XUL y *JavaScript* entre otros, que forman la estructura y el comportamiento de una aplicación.
  - *locale*, archivos DTD y de propiedades entre otros, que contienen cadenas para la traducción de la interfaz de aplicación.
  - *skin*, archivos CSS e imágenes, las cuales definen la apariencia de la interfaz.
- En último lugar, la ruta del archivo a cargar.

Por lo tanto, *chrome://browser/content/browser.xul* carga el archivo *browser.xul* de la sección *content* del componente *browser*.

Explicado qué son las chrome URIs, definimos el archivo *chrome.manifest*. Dentro del archivo se declarará el tipo de material que se encuentra dentro de un paquete chrome, el nombre del paquete y la ruta hacia el mismo. También se define cual es el archivo XUL que se va a fusionar con el archivo del navegador que viene por defecto con *Firefox* (*browser.xul*). El tipo de material que se define puede ser *content*, *locale* y *skin*, definidos anteriormente. Un ejemplo del contenido sería:

```
1 # Registro del tipo de material que contiene la
      extensión.
2 # Tipo de material - Nombre del paquete chrome
      (nombre de la extension) - Ubicación de los
      archivos.
3 content miextension chrome/content/
4 locale miextension chrome/locale/
5 skin miextension chrome/skin/
6 # Fusión del navegador browser.xul de Firefox con
      la interfaz de la extensión.
7 overlay chrome://browser/content/browser.xul
      chrome://miextension/content/menuPrincipal.xul
```

3.1.4. Drag & Drop

Para la realización de la extensión fue necesario un método que permitiera la ordenación de las etiquetas de una forma sencilla e intuitiva para el usuario, y para ello contamos con la funcionalidad de arrastrar y soltar que incluye *Firefox* [17]. Esta funcionalidad le permite al usuario hacer clic y mantener presionado

el botón del ratón sobre un elemento, arrastrarlo a otra ubicación y soltarlo para colocar el elemento allí. Al puntero del ratón le seguirá una representación transparente de lo que se está arrastrando durante la operación. La ubicación de destino puede ser una aplicación diferente. Sitios web, extensiones y aplicaciones XUL pueden hacer uso de esta funcionalidad para personalizar los elementos que pueden ser arrastrados, evaluar la operación, así como especificar el lugar donde los elementos se pueden soltar.

Cuando comienza una operación de arrastre, se pueden proporcionar una serie de datos:

- Los datos que se van a arrastrar, que pueden ser de varios formatos diferentes: texto, enlaces, HTML y XML, archivos, imágenes y nodos de documento.
- La imagen de confirmación sobre la operación de arrastre que aparece junto al puntero durante la operación. Esta imagen puede personalizarse, si no se especifica, se genera una imagen por defecto basándose en el elemento donde se ha pulsado el ratón.
- Los efectos de arrastre que se permiten. Son posibles tres efectos:
  - **copy:** para indicar que los datos que se arrastran se copiarán desde su ubicación actual a la ubicación de destino.
  - **move:** para indicar que los datos que se arrastran serán movidos.
  - **link:** para indicar que se creará algún tipo de relación o conexión entre la ubicación actual y la ubicación de destino.

El manejo del *drag & drop* se realiza a través de diferentes eventos.

- **dragstart:** Se ejecuta sobre un elemento cuando se inicia una operación de arrastre. Durante este evento, un proceso de escucha recogerá información sobre los datos de la operación de arrastre y la imagen que se asocia con ella.
- **dragenter:** Se dispara cuando el ratón se mueve por primera vez sobre un elemento mientras está teniendo lugar una operación de arrastre. Un proceso de escucha de este evento debe indicar si se permite una operación de arrastre o no sobre esta ubicación.
- **dragover:** Este evento se activa cuando el ratón se mueve sobre un elemento cuando está teniendo lugar una operación de arrastre.
- **dragleave:** Este evento se activa cuando el ratón sale de un elemento mientras que está teniendo lugar una operación de arrastre. Los procesos de escucha deben eliminar cualquier resaltado o marcador de inserción que usan para la información sobre el proceso de arrastre.
- **drag:** Este evento se activa en el origen del arrastre, es decir, el elemento donde *dragstart* fue disparado.

- **drop:** El evento se dispara sobre el elemento en el que se soltó el objeto al finalizar la operación de arrastre. Un proceso de escucha se encargará de recuperar los datos que se arrastran e insertarlos en la ubicación donde se soltaron.

- **dragend:** El origen del arrastre recibirá un evento *dragend* cuando la operación se haya completado, tanto si tuvo éxito como si no.

Interfaz *Tree View*

En nuestro caso, al intentar manejar el *drag & drop* de forma primitiva nos encontramos con un problema inesperado. Las etiquetas recogidas con la extensión las representábamos en un árbol de profundidad n-ésima creado por nosotros manualmente, y al intentar aplicar los eventos básicos del *drag & drop* nos encontramos con la imposibilidad de definir un nodo concreto del árbol como destino. Con dichos eventos recogíamos y tratábamos los datos sin problemas, pero a la hora de soltar el arrastre en un nodo destino no había forma de saber en qué nodo se producía la caída. Esto se debe a que todos los nodos del árbol son tratados por igual sin posibilidad de diferenciar entre unos y otros, ya que con estos eventos al producir una caída dentro de un árbol solamente se obtiene la raíz del árbol donde se cuelgan los nodos.

Llegados a este punto tuvimos que recurrir a la interfaz *nsITreeView* [18] ofrecida por *Mozilla Firefox*, la cual mejora el rendimiento comparado con los métodos DOM que usamos al principio para crear el árbol, y nos permitía llevar a cabo el *drag & drop* en un árbol sin complicaciones.

A continuación comentaremos brevemente los principales requisitos para crear una vista de árbol [19, 20]:

- Para la representación básica de un árbol, son necesarias las funciones que devuelvan el número total de filas, el texto de las celdas y la que asigne el objeto vista con el árbol en cuestión.

- En la representación jerárquica de un árbol, aparte de las funciones mencionadas anteriormente, son necesarias las funciones relacionadas con la profundidad de un nodo, las funciones orientadas a los contenedores para indicar si un nodo tiene hijos y la función encargada de controlar la representación de los hijos para cuando se cierra o se abre un nodo contenedor.

- Junto a estas funciones, sería necesario manejar dos matrices de datos, una que contenga los elementos visibles para su representación y otra que contenga todas las posibles relaciones existentes entre los nodos para cuando se abra algún contenedor.

- Para la implementación del *drag & drop*, será necesario una función que pase los nodos cuando comience el arrastre, una que indique si es posible soltar un nodo en un nodo determinado y finalmente una que lleve a cabo la operación caída del arrastre.

3.2. Lenguaje informático

Para la realización de la extensión tuvimos que emplear varios lenguajes informáticos. Existen varios tipos, pero solo comentaremos brevemente los usados en el proyecto.

- **Lenguaje de programación**: es el lenguaje mediante el cual se describe la lógica de un programa. El lenguaje de programación usado en el proyecto es *JavaScript*.
- **Lenguaje de consulta**: es el lenguaje mediante el cual se obtienen unos resultados acordes a la necesidad de un usuario utilizando filtros o condiciones para delimitar y precisar el resultado. Los lenguajes de consulta utilizados han sido XPath y SQL.
- **Lenguaje de marcas**: es un lenguaje utilizado para codificar un documento incorporando etiquetas o marcas que contienen información adicional sobre la estructura del texto o su presentación. Dentro de los distintos campos de aplicación, usaremos uno referido a las tecnologías de Internet, y concretamente a la interfaz de usuario, como es XUL. Dentro de este lenguaje, también se han utilizado puntualmente HTML y CSS, aunque debido a su escaso impacto en el proyecto y al amplio conocimiento que existe sobre ellos, no diremos más que el primero es usado para la representación de las páginas de Internet y el segundo es utilizado para dar estilo a documentos HTML y XML, entre los que se encuentra XUL.

3.2.1. JavaScript

JavaScript [21, 22] es un lenguaje de programación interpretado, lo que significa que está diseñado para ser ejecutado directamente por medio de un intérprete sin necesidad de una compilación previa. Cabe destacar que a pesar del nombre, no está relacionado con el lenguaje de programación Java teniendo semánticas y propósitos diferentes.

Su utilización está principalmente enfocada a aplicaciones web aunque se puede usar en aplicaciones externas como *widgets*. En el uso de aplicaciones web, es utilizado principalmente en el lado del cliente para crear páginas web dinámicas o mejorar la interfaz de usuario.

Sus principales características son:

- **Basado en prototipos**, es un estilo de programación orientada a objetos.
- **Débilmente tipado y dinámico**, lo cual quiere decir que los tipos se asocian con los valores y no con las variables (una variable x puede apuntar a un número y después apuntar a una cadena de texto).
- **Imperativo y estructurado**, imperativo en cuanto a la resolución de un problema detallando los pasos secuenciales necesarios a seguir y estructurado en cuanto al código se divide en bloques o estructuras que pueden o no comunicarse entre si.

Este lenguaje es el que se ha empleado para implementar toda la funcionalidad de este proyecto. Para su inserción en un documento HTML o XML se usa la etiqueta *<script>*. Los scripts pueden estar incrustados en el documento (las etiquetas *<script type="text/javascript">* y *</script>* indican donde comienza y acaba el código) o ir en archivos externos (un ejemplo sería *<script src="myScript.js"></script>*).

DOM

Para la manipulación de los documentos HTML y XML dentro de JavaScript se ha empleado DOM, que no es más que una interfaz de programación de aplicaciones (API) que permite acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML. DOM transforma todos los documentos HTML y XML en un conjunto de elementos llamados nodos, los cuales están interconectados y representan los contenidos de las páginas web y las relaciones entre ellos. La unión de todos los nodos se llama "árbol de nodos" debido a su aspecto. De esta forma, *JavaScript* puede acceder a todos los elementos y atributos de la página y crear y eliminar cualquier elemento y/o atributo. También puede reaccionar a todos los eventos que se produzcan en la página.

3.2.2. XPath

XPath [23] es un lenguaje de consulta que permite construir expresiones que recorren y procesan un documento XML para seleccionar nodos o conjuntos de nodos de dicho documento según una amplia variedad de criterios. Además contiene una amplia biblioteca de funciones estándar, como por ejemplo para valores numéricos, comparación de fecha y hora, valores booleanos...

Este lenguaje se basa en una representación de árbol del documento XML. El elemento superior del árbol se llama el elemento raíz. En XPath, hay siete tipos de nodos: nodos elemento, atributo, texto, espacio de nombres, instrucción de procesamiento, comentarios y documentos. XPath fue definido por el W3C.

XPath en JavaScript

Gracias a la interfaz DOM podemos manipular y acceder a cualquier nodo de los documentos HTML y XML. Pero si el documento llega a ser un poco complejo, acceder a un nodo determinado podría suponer realizar una búsqueda compleja y costosa si no se puede acceder a él de forma directa. Por ejemplo, si se trata de un nodo que no tiene Id, al igual que su padre, y para llegar al nodo hay que hacer varios saltos en la profundidad del árbol y diferenciando al nodo deseado de sus hermanos, la búsqueda se puede volver muy compleja. Es por ello que decidimos recurrir a XPath para realizar estas complejas búsquedas con una simple consulta.

Para poder trabajar en *JavaScript* con XPath [24], *Mozilla* implementa una cantidad razonable de DOM 3 XPath [25]. Lo que significa que las expresiones

XPath se pueden ejecutar en los documentos HTML y XML. La interfaz principal para el uso de XPath es la función *evaluate* del objeto *document*, con la cual podremos evaluar expresiones XPath de forma fácil.

La función *evaluate* se encarga de evaluar las expresiones XPath dentro de un documento XML (incluidos los xHTML). Esta función recibirá 5 parámetros para devolvernos dentro de un objeto *xpathResult* uno o más nodos correspondientes al resultado de nuestra expresión evaluada.

```
1 var xpathResult = document.evaluate(xpathExpression,
      contextNode, namespaceResolver, resultType,
    result);
```

Las descripciones de los parámetros de la función son:

- **xpathExpression**: una cadena que contiene la expresión XPath para ser evaluada.
- **contextNode**: un nodo en el documento contra el cual la *XPathExpression* debe ser evaluada, incluyendo todos y cada uno de sus nodos secundarios. El nodo *document* suele ser el más empleado debido a que desde él se puede acceder a cualquier otro nodo.
- **namespaceResolver**: en este parámetro se indica una función que permitirá resolver el prefijo del *namespace* que contenga nuestra expresión. Esta función puede ser:
  - *Created*, esta función debe emplearse siempre de modo virtual con el método *createNSResolve* de *xpathEvaluator*.
  - *null*, para documentos HTML o cuando no se usan *namespaces*.
  - *Función definida por el usuario*, cualquier funcionalidad que se desee implementar.
- **resultType**: una constante que especifica el tipo deseado en el que se quiere recibir el resultado, el más usado es ANY_TYPE.
  - *ANY_TYPE (0)*: de forma automática decide el mejor tipo recomendado, en caso de devolver un conjunto de nodos devolverá siempre una lista no ordenada de nodos.
  - *NUMBER_TYPE (1)*: el resultado contiene un número, generalmente usado para la función *count()* de XPath.
  - *STRING_TYPE (2)*: el resultado devuelto es un String, usado para devolver el valor del nodo.
  - *BOOLEAN_TYPE(3)*: devuelve un tipo booleano, usado para comprobar la existencia de un nodo (*not()*).

- *UNORDERED_NODE_ITERATOR_TYPE(4)*: devuelve un listado de nodos que no tienen por qué estar ordenados de igual forma en que aparezcan en el documento XML o HTML.
- *ORDERED_NODE_ITERATOR_TYPE(5)*: un listado de nodos colocados de igual forma que su aparición en el documento.
- *UNORDERED_NODE_SNAPSHOT_TYPE(6)*: devuelve una copia de los nodos resultantes de la expresión, no necesariamente ordenados según su aparición.
- *ORDERED_NODE_SNAPSHOT_TYPE(7)*: devuelve una copia de los nodos ordenados según su aparición.
- *ANY_UNORDERED_NODE_TYPE(8)*: devuelve un solo nodo que no tiene por qué ser el primer nodo del documento.
- *FIRST_ORDERED_NODE_TYPE(9)*: devuelve el primer nodo del resultado de la expresión.

- **result**: este parámetro está destinado para reutilizar el objeto *xpathResult*, usando *null* crearemos uno nuevo.

Cuando el tipo de resultado deseado es un número, una cadena o un booleano, para obtener el valor devuelto de la expresión se necesitan las propiedades *numberValue*, *stringValue* y *booleanValue* respectivamente.

La diferencia entre el listado *Iterator* y el listado *Snapshot* es que en el primero se nos está devolviendo una referencia a los nodos, de forma que si estos cambian también cambiarán en nuestro resultado. Mientras que en el segundo se nos devuelve una copia exacta, de manera que si cambiaran los nodos originales los almacenados en el *snapshot* no sufrirán cambios.

Para recorrer el *iterator* se utiliza la función *iterateNext(),* de forma que cuando se haya recorrido todo el listado la función devolverá *null*. Para recorrer el *snapshot* se utiliza la función *snapshotItem(itemNumber)*, donde *itemNumber* es el índice del nodo a recuperar. Para obtener el número total de nodos se usa la propiedad *snapshotLength*.

Cuando el tipo de resultado es el primer nodo encontrado, para acceder a él se utiliza la propiedad *singleNodeValue*, que devolverá *null* si está vacío.

3.2.3. SQL

El lenguaje de consulta estructurado o SQL [26, 27, 28] es un lenguaje de acceso a bases de datos relacionales que permite efectuar consultas con el fin de recuperar información de interés así como hacer cambios en la propia base de datos. Su ámbito de aplicación incluye inserción de datos, consulta, actualización y eliminación, esquema de creación y modificación, y el control de acceso a datos.

SQL es un lenguaje estándar[8] de comunicación con bases de datos que permite realizar operaciones básicas de forma universal. Es un lenguaje normalizado

[8] SQL es un estándar ANSI

que nos permite trabajar con cualquier tipo de lenguaje (PHP, *JavaScript*, ...) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL, SQLite, ...). El hecho de que sea estándar no implica que sea idéntico para todas las base de datos, ya que determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Sintaxis y comandos

La mayor parte de las acciones que se deben llevar a cabo en una base de datos se hacen con sentencias SQL. Estas sentencias están compuestas por comandos, cláusulas, operadores y funciones de agregado. Un ejemplo de una sentencia SQL es la siguiente, la cual selecciona todos los registros almacenados en la tabla Personas.

```
1  SELECT * FROM Personas
```

Dependiendo de las tareas, estas sentencias se pueden clasificar en tres grupos principales: el lenguaje de definición de datos (DDL), el lenguaje de manipulación de datos (DML) y el lenguaje de control de datos (DCL).

Los DDL permiten crear y definir nuevas bases de datos, campos e índices. Las sentencias DDL más importantes son:

- *CREATE*: utilizado para crear nuevas tablas, campos e índices.
- *DROP*: empleado para eliminar tablas e índices.
- *ALTER*: utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Los DML permiten actualizar datos y generar consultas para ordenar, filtrar y extraer datos de la base de datos. Las sentencias DML son:

- *SELECT*: utilizado para consultar los datos de una base de datos.
- *UPDATE*: utilizado para modificar los datos en una base de datos.
- *DELETE*: utilizado para eliminar los datos de una base de datos.
- *INSERT*: utilizado para cargar datos nuevos en una base de datos.

Los DCL se ocupan de los aspectos de control de datos, controlando los privilegios de los usuarios y controlando que las transacciones se completen o se deshagan hasta un estado anterior.

- *GRANT*: concede privilegios de acceso a usuarios.
- *REVOKE*: suprime privilegios de acceso a usuarios.

- *START TRANSACTION*, *BEGIN* o *BEGIN TRANSACTION*: utilizado para marcar el inicio de una transacción, la cual se completará del todo o no se hará nada.
- *SAVE TRANSACTION* o *SAVEPOINT*: utilizado para guardar el estado de la base de datos en el punto actual en la transacción
- *COMMIT*: utilizado para finalizar la transacción actual realizando todos los cambios de datos de manera permanente.
- *ROLLBACK*: utilizado para abortar la transacción actual provocando que todos los cambios realizados desde el último *COMMIT* o *ROLLBACK* sean desechados.

3.2.4. XUL

XUL [29] es un lenguaje basado en XML utilizado para describir y crear interfaces de usuario, creado inicialmente para desarrollar los productos de *Mozilla*. Gecko es el motor del navegador *Mozilla* y el encargado de interpretar las descripciones XUL. Debido a su sencillez y portabilidad, la interfaz completa del navegador *Mozilla* está implementada en este lenguaje. En XUL es posible crear una interfaz usando un lenguaje de marcado, definir su apariencia con hojas de estilo CSS y usar *JavaScript* para agregar funcionalidades. Además tiene un conjunto extenso de componentes gráficos como pueden ser controles de entrada (TextBox, CheckBox, etc), diálogos emergentes (pop-up), árboles y barras de herramientas entre otros.

En resumen, XUL puede usarse para crear interfaces multiplataformas, multidispositivos y ligeras. Es por ello que podemos encontrar aplicaciones XUL en extensiones de *Firefox* (agregan funcionalidades al navegador), aplicaciones independientes (no es necesario el navegador para ejecutarlas al tener su propio ejecutable) y aplicaciones remotas (se encuentran en un servidor web y se ejecutan remotamente).

Sintaxis

Un archivo XUL puede tener cualquier nombre, pero debería tener una extensión *.xul. Un ejemplo de archivo XUL tiene la siguiente estructura:

```
1 <?xml version="1.0"?>
2 <?xml-stylesheet href="chrome://global/skin/" type="
      text/css"?>
3 <!DOCTYPE window SYSTEM "chrome://miExtension/locale
      /ejemplo.dtd">
4 <window
5   id="ejemplo-window"
6   title="Mi ejemplo"
7   orient="horizontal"
```

```
8     xmlns="http://www.mozilla.org/keymaster/gatekeeper
          /there.is.only.xul">
9   <script type="application/x-javascript" src="ejemplo
        .js"/>
10  <!-- Otros elementos irán aquí -->
11  </window>
```

XUL es XML, y como todos los archivos XML, debe empezar con la declaración estándar de XML. A continuación, estarían las referencias a los CSS y DTD utilizados. No existen hojas de estilo implícitas para XUL, por lo que siempre debe existir una declaración de hoja de estilo asociada. Posteriormente están las etiquetas que permiten cargar todos los scripts que utilizaremos. Luego vemos que el elemento principal de este ejemplo es *<window>*, que será el elemento que contenga al resto de elementos. Ya dentro de esa ventana podremos ir agregando los componentes gráficos que deseemos, tales como cajas de texto, árboles, botones, etc. Es así de sencillo como se pueden llegar a crear interfaces de usuario complejas.

Algunas reglas de sintaxis a tener en cuenta también son:

- Los elementos en XUL y sus atributos deben introducirse en minúsculas, ya que XML distingue entre mayúsculas/minúsculas (a diferencia de HTML).
- Los valores de los atributos en XUL, deben colocarse entre comillas, aunque sean números.
- Todos los componentes interactivos deben tener etiquetas de cierre (*<etiqueta></etiqueta>* o *<etiqueta/>*) para ser bien formados.
- Todos los atributos deben tener un valor.
- Los archivos XUL por lo general se dividen en cuatro ficheros, uno para la disposición de los elementos, otro para la definición del estilo, otro para declarar las entidades (usadas en las localizaciones para definir varios idiomas) y otro para los scripts. Además se pueden tener archivos para las imágenes o para datos específicos de una plataforma.

3.3. Almacenamiento de datos

Para llevar a cabo la realización de la extensión tuvimos la necesidad de almacenar cierta información para su posterior acceso en futuras consultas. Como requisitos principales necesitábamos que la información almacenada se guardara en algún fichero en el propio equipo sin necesidad de instalar ningún programa externo para ello, y que ese almacenamiento fuera multiplataforma al igual que el navegador. Es por ello que necesitamos usar una base de datos en nuestro proyecto, siendo SQLite la que cumplía con nuestros requisitos previos.

Cabe destacar que como alternativas a SQLite teníamos CSV y mongoDB que también cumplían con los requisitos. CSV es un fichero de texto plano utilizado para representar datos en forma de tablas en las que las columnas se separan por comas y las filas por saltos de línea. Y mongoDB es un sistema de base de datos NoSQL orientada a documentos, que guarda los datos en documentos tipo JSON con un esquema dinámico llamado BSON. Pero nos decantamos por SQLite porque a diferencia de las otras dos alternativas controlar la integridad de los datos era mucho más sencillo, ya que la información que almacenamos para representar un árbol jerárquico se basa en las relaciones entre los datos almacenados.

3.3.1. Base de datos

Para manipular las bases de datos [28] es necesario un sistema de gestión de bases de datos, conocido como DBMS, que es un tipo de software muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Nosotros usaremos un sistema de gestión de bases de datos relacional, conocido como RDBMS. Este sistema está basado en el modelo relacional, el cual está fundamentado en el uso de relaciones y donde el lugar y la forma de almacenamiento no tiene relevancia. En este modelo cada relación se trata como si fuera una tabla compuesta por filas y columnas. Tal y como ya comentamos en la sección 3.2.3, el lenguaje más habitual para construir las consultas en estas bases de datos es SQL.

Las bases de datos relacionales pasan por un proceso al que se le conoce como normalización de una base de datos,el cual es entendido como el proceso necesario para que una base de datos sea utilizada de manera óptima. Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Evitar problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

3.3.2. SQLite

SQLite [30, 31] es un sistema de gestión de bases de datos relacional, contenido en una pequeña librería de aproximadamente 500KB programada en lenguaje C. Es un proyecto de dominio público creado por D. Richard Hipp. Encapsula toda la base de datos en un único fichero estándar dentro del propio equipo. Su potencia se basa fundamentalmente en la simplicidad, lo que hace que no sea una buena solución en entornos de tráfico muy elevado y/o alto acceso concurrente a datos. En su versión 3, SQLite soporta bases de datos de hasta 2 TeraBytes de tamaño. También permite la inclusión de campos tipo BLOB, los cuales son elementos utilizados en las bases de datos para almacenar datos de gran tamaño que cambian de forma dinámica. Generalmente estos datos suelen ser archivos

multimedia. SQLite es utilizado en una gran variedad de aplicaciones, entre las que se encuentra *Mozilla Firefox*, el cual usa SQLite para almacenar, entre otros, las cookies, los favoritos, el historial y las direcciones de red válidas.

Características, ventajas y desventajas

Las características de SQLite son:

- **No posee configuración**: no necesita ser instalado ni gestionar un servidor.
- **Multiplataforma**: puede ser ejecutado en diferentes sistemas operativos ya que al estar la base de datos condensada en un único fichero facilita la portabilidad de los datos, teniendo solamente como restricción el espacio de disco en el equipo.
- **Registros de longitud variable**: SQLite emplea registros de longitud variable de forma tal que se utiliza el espacio en disco que es realmente necesario en cada momento, a diferencia de otros motores que asignan una cantidad fija de espacio. Esto tiene como resultado un pequeño archivo de base de datos y una optimización en la velocidad al haber menos información desperdiciada que leer y recorrer.

Las ventajas de SQLite son:

- **Tamaño**: SQLite tiene una pequeña memoria y una única biblioteca es necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas.
- **Rendimiento de base de datos**: SQLite realiza operaciones de manera rápida y eficiente, siendo más rápido que MySQL y PostgreSQL.
- **Portabilidad**: es multiplataforma y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración.
- **Estabilidad**: SQLite es compatible con ACID, reunión de los cuatro criterios de Atomicidad, Consistencia, Aislamiento y Durabilidad.
- **SQL**: implementa un gran subconjunto del lenguaje estándar SQL.
- **Interfaces**: cuenta con diferentes interfaces del API, las cuales permiten trabajar con C++, PHP, Perl, Python, Ruby, Tcl, groovy, etc.
- **Coste**: SQLite es de dominio público, y por tanto, es libre de utilizar para cualquier propósito sin coste y se puede redistribuir libremente.

A pesar de sus potentes ventajas, tiene las siguientes limitaciones:

- **Limitaciones en *Where***: esta limitación está dada por el soporte para clausuras anidadas.

- **Falta de Clave Foránea**: ignora las claves foráneas, es decir, cuando se realiza la creación de la tabla permite el uso de claves foráneas aunque no realizará ninguna comprobación. En las últimas versiones de SQLite las claves foráneas ya están soportadas, aunque por defecto vienen desactivadas por la compatibilidad hacia atrás. Para mantener la integridad en la base de datos, si la clave foránea está desactivada se pueden usar *TRIGGER*.

API para Mozilla Firefox

Mozilla Firefox ofrece una API para bases de datos llamada Storage [32]. El procedimiento general para usarlo es:

1. Abrir una conexión a la base de datos de su elección.
2. Crear las sentencias para ejecutar la conexión.
3. Enlazar tantos parámetros a las sentencias como sea necesario.
4. Ejecutar las sentencias.
5. Reiniciar las sentencias.

La primera inicialización del servicio de almacenamiento debe hacerse en el hilo principal, produciéndose un error si se inicializa en otro hilo.

```
1 var file = Components.classes["@mozilla.org/file/
      directory_service;1"].getService(Components.
      interfaces.nsIProperties).get("ProfD",
      Components.interfaces.nsIFile);
2 file.append("prb.sqlite");
3 var storageService = Components.classes["@mozilla.
      org/storage/service;1"].getService(Components.
      interfaces.mozIStorageService);
4 var mDBConn = storageService.openDatabase(file);
```

Existen dos formas de crear una sentencia. Si la sentencia no devuelve nada se utiliza la función *executeSimpleSQL*, y si esperamos que devuelva algo se utiliza la función *createStatement*.

```
1 mDBConn.executeSimpleSQL("CREATE TABLE prb (a
      INTEGER)");
2 var statement = mDBConn.createStatement("SELECT *
      FROM prb WHERE a = 5");
```

Se pueden asignar parámetros por separado en lugar de definirlos directamente en la sentencia. Los parámetros no asignados se se interpretarán como *NULL*.

```
var statement = mDBConn.createStatement("SELECT *
    FROM prb WHERE a = ?1 AND b > ?2");
statement.bindUTF8StringParameter(0, "hola");
statement.bindInt32Parameter(1, 1234);
```

Para ejecutar una sentencia se suele utilizar la función *executeStep*. Esta función permite enumerar todas las filas resultantes que la sentencia ha producido y avisará cuando ya no haya más resultados.

```
while (statement.executeStep()) {
  var value = statement.GetInt32(0);
  //uso del valor obtenido
}
```

Cuando no se esperan datos al ejecutar la sentencia se puede usar la función *execute*, que itera una vez la sentencia y la reinicia.

```
var statement = mDBConn.createStatement("INSERT INTO
    my_table VALUES (?1)");
statement.bindInt32Parameter(52);
statement.execute();
```

Es importante reiniciar sentencias que ya no van a ser usadas más. Las sentencias de escritura no reiniciadas mantendrán bloqueadas las tablas e impedirán a otras sentencias acceder a ellas. Las sentencias de lectura no reiniciadas impedirán la escritura.

```
var statement = connection.createStatement(...);
try {
  // uso de la sentencia...
} finally {
  statement.reset();
}
```

Dentro de la API también se ofrecen funciones para comenzar y finalizar transacciones. Si no se usan transacciones de modo explícito, se creará una implícita para cada sentencia. El uso de transacciones puede mejorar notablemente el rendimiento cuando se realicen múltiples sentencias en una transacción. También se puede ejecutar *BEGIN TRANSACTION* y *END TRANSACTION* directamente como sentencias SQL.

3.4. Integración de las tecnologías y despliegue del entorno

Para la realización del proyecto, tal y como se ha dicho anteriormente, se decidió realizar una extensión para el navegador *Mozilla Firefox*. Para el desarrollo de dicha extensión es necesario crear una estructura básica que han de cumplir todas las extensiones. Dentro de esa estructura son imprescindibles el fichero que describe los aspectos de instalación, el fichero que registra el contenido creado de dicha extensión y el contenido propio de la extensión como son las interfaces creadas y la funcionalidad implementada.

Las interfaces creadas de la extensión fueron escritas en XUL, que es lenguaje usado en *Mozilla Firefox* para crear las interfaces de usuario. Éstas fueron complementadas con hojas de estilo CSS. Toda la funcionalidad implementada fue escrita en *JavaScript*, incluyendo el lenguaje SQL para la manipulación de los datos almacenados y el lenguaje XPATH para el acceso a determinados nodos de los documentos HTML y XML. Dentro de la funcionalidad implementada, se utilizó la interfaz *Tree View* ofrecida por *Mozilla Firefox*, la cual facilita la implementación de la funcionalidad *drag & drop* usada para la ordenación de los datos.

Cabe mencionar la recomendación de disponer previamente de algunas herramientas para facilitar el desarrollo de la extensión. Para ello utilizamos algunos complementos existentes en el navegador, los cuales fueron:

- **DOM Inspector**: permite inspeccionar, navegar y editar el árbol DOM de los documentos, como son páginas web y ventanas XUL.
- **Firebug**: permite editar, depurar y monitorear el CSS, HTML y *JavaScript* en vivo en cualquier página web.
- **SQLite Manager**: permite administrar cualquier base de datos SQLite.

Capítulo 4

Diseño

En este capítulo se describirá el diseño que se ha realizado de la extensión, a partir de lo estudiado en el estado del arte del capítulo 2 y de las tecnologías del capítulo 3, para comprender su funcionamiento sin entrar en profundidad en los detalles. Empezaremos indicando los requisitos que deberá cumplir la extensión y acabaremos detallando un diseño de alto nivel de la misma explicando cómo funciona e interactúan las diferentes partes implicadas.

4.1. Requisitos del sistema

Para la realización de este proyecto se plantearon una serie de requisitos a cumplir, los cuales vienen dados por los objetivos puestos al inicio del proyecto y lo visto en los capítulos anteriores. A continuación enumeraremos los requisitos:

1. El proyecto consistirá en una extensión para el navegador *Mozilla Firefox*, que permita al usuario establecer un sistema de clasificación jerárquico sobre sistemas basados en clasificación por etiquetado para facilitar sus búsquedas. Con ello se persigue superar las limitaciones que por separado presentan los sistemas de clasificación por etiquetado y los sistemas de clasificación jerárquico, combinándolos en lo que se ha denominado sistema de clasificación por etiquetado jerárquico.

2. Tal y como comentamos en el capítulo 1 la extensión funcionará en el portal *Delicious*, aunque podrá funcionar en otros sitios web agregando pequeños módulos adaptados para esos sitios concretos. Esto se debe a que se quiere desarrollar una herramienta genérica que pueda ser utilizada en diferentes aplicaciones y no sea específica para una determinada aplicación.

3. La extensión estará basada en la arquitectura "Modelo Vista Controlador" y por tanto dividida en tres módulos independientes para facilitar su desarrollo y adaptarse mejor a modificaciones. Un módulo será el que contenga una representación de los datos que maneja la extensión, Modelo, otro será

el referente a las interfaces de usuario, Vista, y el último será el encargado de gestionar las acciones que realice el usuario haciendo de intermediario entre los anteriores, Controlador.

4. La interfaz gráfica será implementada en XUL, por ser el lenguaje utilizado por *Mozilla* para crear interfaces de usuario, mientras que la funcionalidad será implementada en *JavaScript*, por ser un lenguaje orientado a aplicaciones web, como hemos podido ver en el capítulo 3.

5. La información estará almacenada en una base de datos gestionada por SQLite, debido a las características y ventajas vistas en la sección 3.3.2, y estará contenida en un único fichero.

6. La extensión funcionará para un único usuario ya que consideramos el complemento de uso individual y personal. Individual porque consideramos que no debería haber más de un usuario por equipo, si hubiera más de un usuario, consideramos que tendrían sesiones diferentes en el mismo equipo y por tanto podrían coexistir las extensiones ya que estarían en perfiles diferentes sin compartir información. Personal porque serán las etiquetas del usuario las que ordenará a su criterio y las que podrá editar si lo considerase oportuno, carecería de sentido organizar las etiquetas de un usuario ajeno a nosotros porque no podríamos gestionar ni controlar los cambios que el usuario ajeno pudiera hacer en sus etiquetas.

7. La ordenación de las etiquetas del portal se realizará en una ventana mediante mecanismos de arrastre y caída.

8. Si en algún momento se elimina alguna etiqueta ya recogida por la extensión y ésta contuviera etiquetas jerárquicamente, se eliminarán todas las relaciones que tuviera pasando las etiquetas hijo a ser nodo raíz sin tener dependencias.

9. La representación del nuevo método de clasificación será opcional, pudiendo dejar el sistema propio del portal. No se forzará al usuario a usar nuestro método de clasificación sino que tendrá la opción de escoger el que más le interese en cada momento.

10. Las imágenes que se puedan usar en las interfaces de usuario serán creadas por nosotros o de alguna web como *OpenClipArt* donde las imágenes no tienen ningún tipo de derecho para su uso libre.

4.2. Diseño de la extensión

En el diseño previo de esta extensión describiremos las partes implicadas, las cuales veremos en la figura 4.1. En primer lugar nos encontraremos con un usuario que interactúa con la propia extensión del navegador (que previamente se ha instalado), la cual se comunica con la página web de *Delicious* mostrada en el navegador. Si fuera necesaria la acción del usuario, la extensión se comunicaría

con el propio usuario para indicarle las acciones necesarias. Toda la información que maneja la extensión será almacenada en una base de datos local con la que se comunicará la extensión.

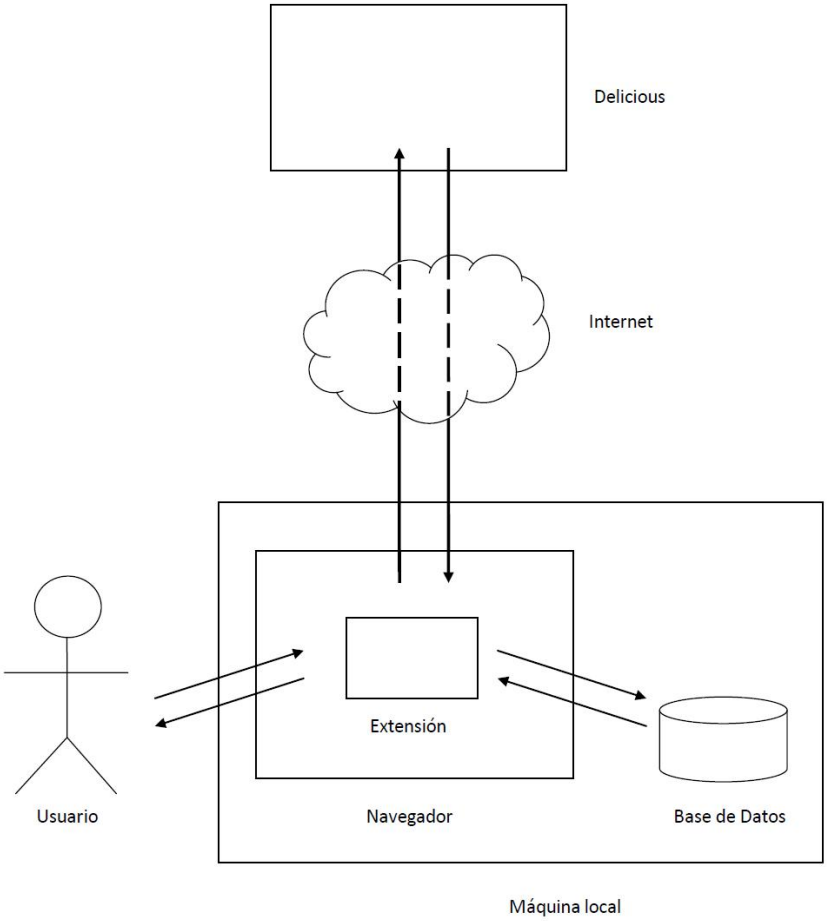

Figura 4.1: Arquitectura de la extensión

Para comprender toda la lógica que tiene la extensión utilizaremos diagramas de flujo, los cuales podremos ver en las siguientes figuras. La funcionalidad de actualizar etiquetas, ver figura 4.3, verifica primero que se haya almacenado un usuario, después que se está identificado en el portal de *Delicious* y finalmente que se muestren todas las etiquetas. En caso contrario muestra unos mensajes de alerta indicando qué debe hacer el usuario para poder actualizar las etiquetas.

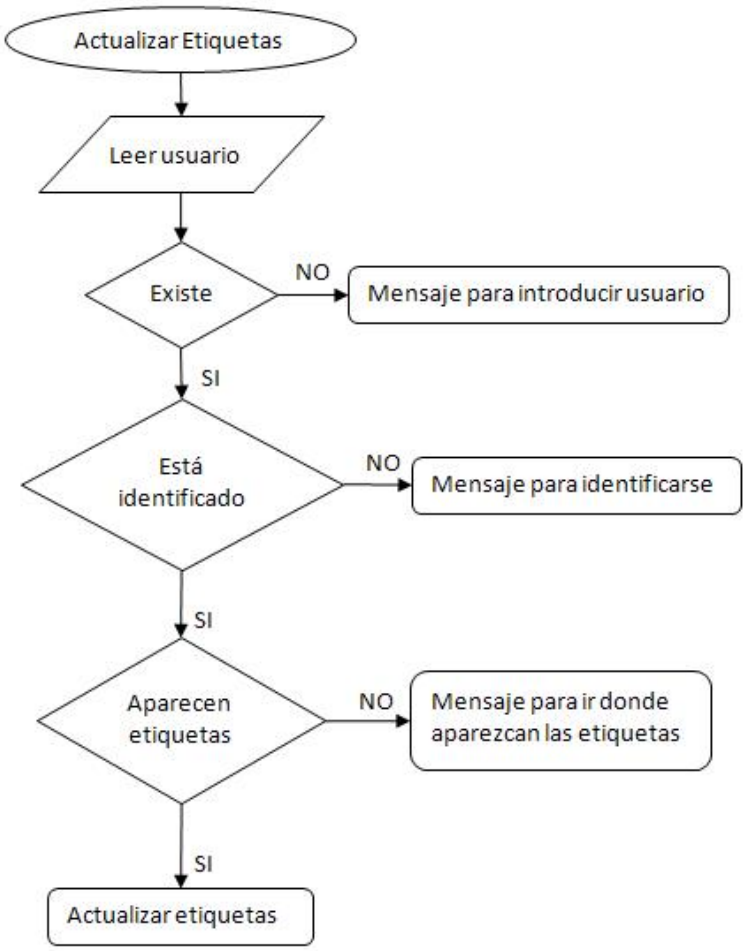

Figura 4.2: Diagrama de flujo - Actualizar etiquetas

Para mostrar las etiquetas de forma jerárquica, ver figura 4.3, se leen las tablas que se usan para almacenar las etiquetas y la estructura jerárquica. Si no hay etiquetas almacenadas se muestra un único nodo del árbol con un mensaje indicando que no hay etiquetas almacenadas. Si existen etiquetas pero no hay almacenada ninguna estructura jerárquica, se muestran según el orden en el que están almacenadas las etiquetas. En caso de sólo existir información del orden de las etiquetas se muestran en dicho orden. Si existiera la información completa del orden de las etiquetas y las relaciones existentes entre ellas, se mostrarán jerárquicamente. Cabe destacar que si por una actualización posterior de las etiquetas se agregan nuevas, una vez representadas las etiquetas previas

de forma jerárquica, se añadirían las nuevas al final sin dependencia alguna con las anteriores.

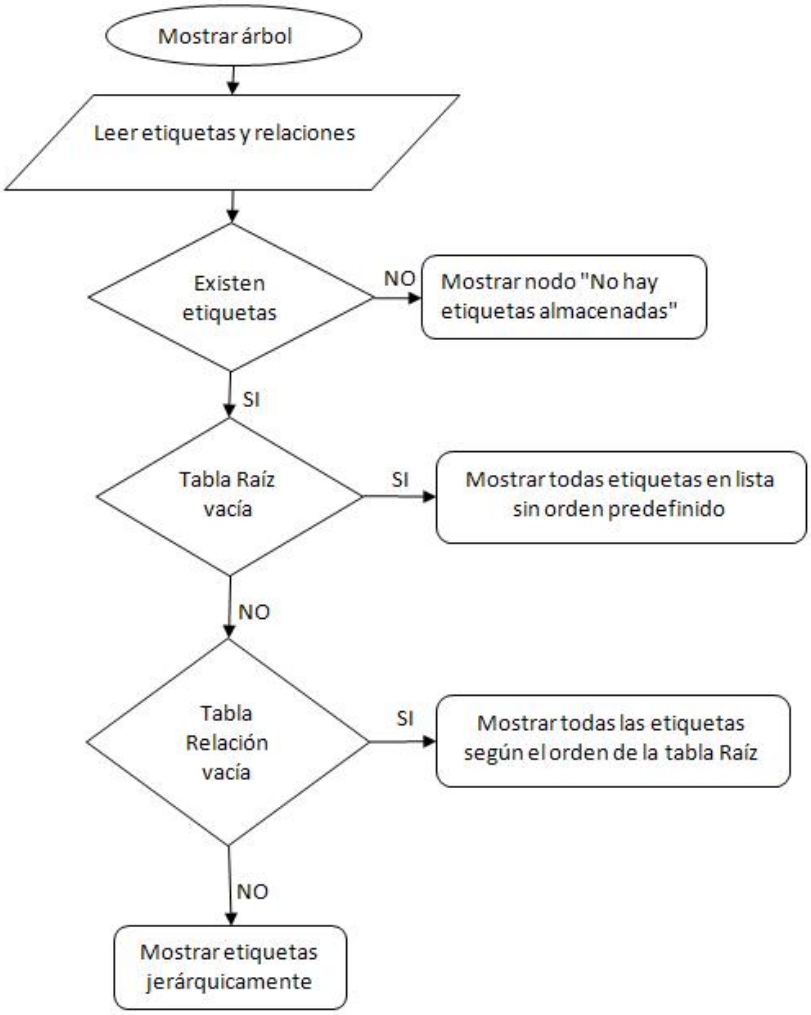

Figura 4.3: Diagrama de flujo - Mostrar árbol jerárquico

Para el diseño de la interfaz gráfica se estableció como requisito que fuera simple y lógica, de forma que el usuario tenga todo accesible rápidamente sin necesidad de realizar una compleja exploración y evitar que pudiera haber posibles confusiones que le puedan llevar a error. En la figura 4.4 podemos ver una captura de pantalla que presenta el diseño de la interfaz gráfica.

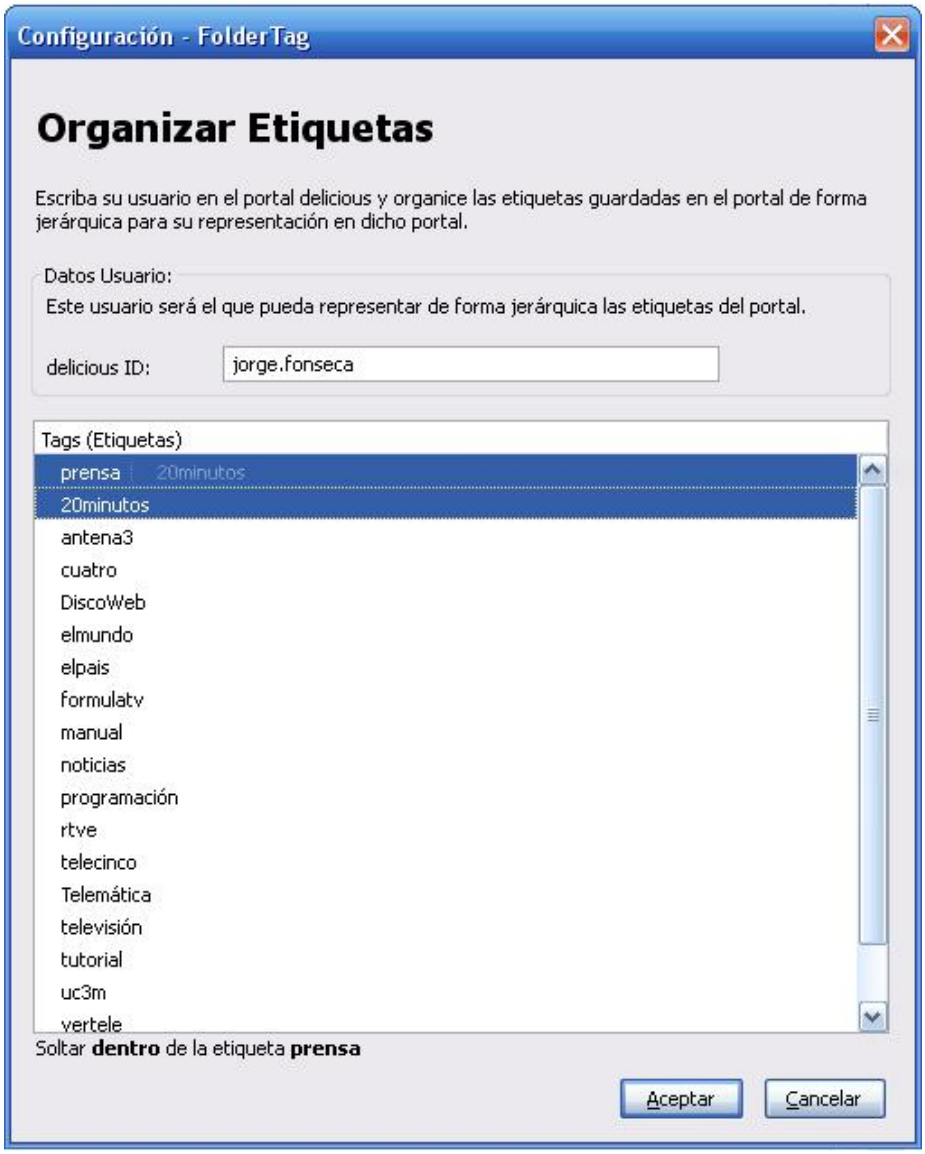

Figura 4.4: Interfaz gráfica de la ventana de configuración

Para la representación de las etiquetas de forma jerárquica se hizo necesario realizar un diseño de la base de datos que almacenase toda la información necesaria para ello. Podemos ver en la figura 4.5 el esquema de la base de datos empleado, y en la figura 4.6 un ejemplo de cómo se almacenarían las etiquetas de una estructura jerárquica.

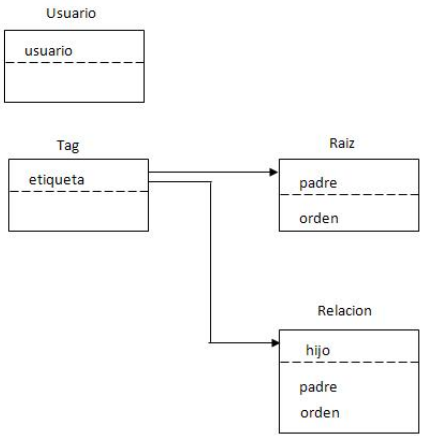

Figura 4.5: Esquema base de datos

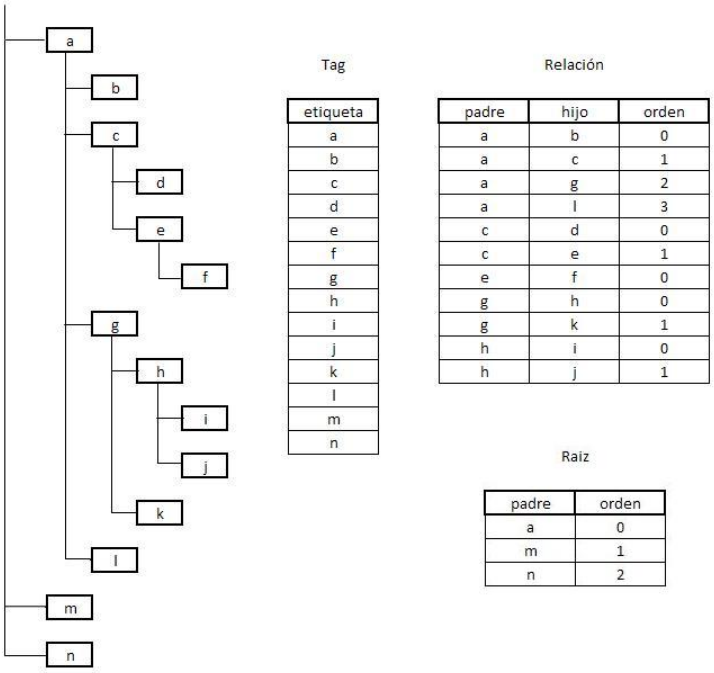

Figura 4.6: Ejemplo de etiquetas almacenadas

A continuación explicaremos en detalle las funciones de cada tabla:

- Usuario. Esta tabla será la que almacene el identificador del usuario en el portal *Delicious*. Contendrá una única columna de tipo texto para almacenar el usuario. Esta tabla no tiene relación con el resto de tablas porque sólo se usa para el usuario de una única aplicación, pero en futuras mejoras podrá llegar a almacenar más usuarios, si se generalizase la extensión a más aplicaciones web, llegando a tener así relación con las otras tablas.

- Tag. Esta tabla será la que almacene todas las etiquetas del portal *Delicious*. Contendrá una única columna de tipo texto para almacenar las etiquetas. Con la futura mejora de usar la extensión para más aplicaciones será necesario relacionar la etiqueta con el usuario, o en su defecto con la aplicación, agregando un campo más.

- Raiz. Esta tabla será la que almacene aquellas etiquetas que están en el nivel 0 del árbol, es decir, las que están en el nodo raíz del árbol. Contendrá una columna de tipo texto para almacenar el nombre de la etiqueta, y otra columna de tipo entero indicando el orden que tienen dichas etiquetas (sus posibles valores irán del 0 en adelante de forma consecutiva).

- Relacion. Esta tabla será la que almacene las relaciones jerárquicas de las etiquetas. Contendrá una columna de tipo texto para almacenar la etiqueta hijo que tiene una relación jerárquica, otra columna de tipo texto para almacenar el padre de la etiqueta hijo y otra columna de tipo entero para indicar el orden que ocupa dentro de los hijos de un padre (sus posibles valores irán del 0 en adelante de forma consecutiva).

Capítulo 5

Implementación

En este capítulo se va a describir con detalle la implementación seguida para desarrollar la extensión *FolderTag*. En concreto, se describirá la estructura de la extensión, los ficheros necesarios para la instalación, los ficheros de contenido utilizados para implementar la interfaz gráfica de usuario y la lógica de la aplicación, los ficheros de configuración regional y los ficheros de aspecto. Por último, realizaremos unas observaciones generales que tuvimos en cuenta durante el desarrollo de la extensión.

5.1. Estructura de la extensión

Como ya se ha comentado en la sección 3.1.3, el paquete que contiene la extensión debe seguir una estructura básica contenida en un archivo con extensión XPI. Este fichero no es más que un archivo ZIP comprimido, renombrando la extensión del archivo a XPI. Este paquete sigue la misma estructura básica que cualquier extensión para *Mozilla Firefox*, permitiendo su instalación de manera sencilla. El paquete de esta extensión se llama *folderTag.xpi* y está estructurado de la siguiente manera:

folderTag.xpi

- chrome.manifest
- intall.rdf
- chrome
  - content
    - about.xul
    - firefox-overlay.xul
    - option.xul
    - firefox-overlay.js
    - folderTag.js

        - folderTagTreeView.js
        - option.js
        - overlay.js
        - sqlite.js
    - locale
        - es-ES
            - about.dtd
            - overlay.dtd
            - option.properties
            - overlay.properties
    - skin
        - images
            - folderTag16.png
            - folderTag32.png
            - folderTag64.png
            - menu_icon.png
            - refresh.png
        - about.css
        - overlay.css

Como podemos ver en la estructura detallada, en el primer nivel se encuentran los dos archivos de configuración y el directorio *chrome*, el cual contiene a su vez otros tres directorios: *content* para los archivos de interfaz de usuario y la funcionalidad implementada, *locale* para los archivos de configuración de idiomas y *skin* para los estilos e imágenes utilizadas.

Cabe mencionar que todos estos ficheros han sido desarrollados desde cero a excepción del fichero *sqlite.js*. Este fichero se tomó prestado de una fuente de Internet, la cual ya no se encuentra disponible, donde se explicaba cómo gestionar la base de datos SQLITE con *JavaScript* para una extensión de *Firefox*. Las imágenes usadas también han sido creadas, salvo dos de ellas que fueron cogidas de una web de imágenes sin derechos de autor (*menu_icon.png* y *refresh.png*). A continuación iremos explicando las distintas partes del paquete XPI con más detalle.

5.2. Ficheros de instalación

Tal y como se ha comentado, para que una extensión se pueda instalar en el navegador *Mozilla Firefox*, son imprescindibles los ficheros *install.rdf* y *chorme.manifest*. El primero describe los aspectos de instalación, tales como las versiones del navegador compatibles, el autor y la propia extensión, y el segundo registra el contenido creado en la extensión. Estos ficheros deben estar en el nivel raíz del paquete XPI. A continuación, vemos ambos ficheros y detallaremos las principales características.

```
<?xml version="1.0" encoding="UTF-8"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <em:id>foldertag@jfonseca.es</em:id>
    <em:type>2</em:type>
    <em:name>FolderTag - Delicious</em:name>
    <em:version>1.0</em:version>
    <em:creator>Jorge Fonseca</em:creator>
    <em:contributor></em:contributor>
    <em:description>Este complemento permite visualizar las
        etiquetas (tags) del portal Delicious de forma
        jerárquica. Para ello el usuario mediante una ventana
         relacionará las diferentes etiquetas, permitiendo
        guardar dicha relación en el propio equipo.</
        em:description>
    <em:optionsURL>chrome://folderTag/content/option.xul</
        em:optionsURL>
    <em:aboutURL>chrome://folderTag/content/about.xul</
        em:aboutURL>
    <em:iconURL>chrome://folderTag/skin/images/folderTag32.
        png</em:iconURL>
    <em:icon64URL>chrome://folderTag/skin/images/folderTag64
        .png</em:icon64URL>
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id
            > <!-- Firefox -->
        <em:minVersion>3.6</em:minVersion>
        <em:maxVersion>41.*</em:maxVersion>
      </Description>
    </em:targetApplication>
  </Description>
</RDF>
```

Como podemos ver, en el fichero *install.rdf* especificamos los metadatos de la extensión, tales como el identificador, el nombre, la versión, el autor, la descripción y la ubicación de algunas partes como son el icono de la extensión y las interfaces gráficas de opciones e información. Debido a que todos los complementos de *Mozilla* usan la misma estructura es necesario especificar para qué aplicación es la extensión y las versiones de la aplicación entre las que funciona nuestro complemento. Esta información se incluye dentro de la etiqueta *targetApplication*, poniendo en nuestro caso el GUID de *Firefox* para indicar la aplicación a la que va dirigida la extensión y las versiones mínimas y máximas del mismo en las que funciona nuestra extensión.

```
content   folderTag                  chrome/content/
skin      folderTag   classic/1.0    chrome/skin/
locale    folderTag   es-ES          chrome/locale/es-ES/

overlay   chrome://browser/content/browser.xul
    chrome://folderTag/content/firefox-overlay.xul
```

Tal y como se observa en el código, y como ya mencionamos en la sección 3.1.3, en el fichero *chrome.manifest* registramos los nombres de los paquetes y las localizaciones físicas de los mismos. En las tres primeras líneas se declara la ubicación de los diferentes tipos de material que usamos en nuestra extensión. En la última línea se indica al navegador que fusionamos la ventana principal del navegador, *browser.xul*, con el fichero *firefox-overlay.xul* que es la interfaz de la extensión incorporada al navegador.

Una vez comentados ambos ficheros, explicaremos a continuación todo el contenido de la extensión que se encuentra dentro del directorio *chrome*, el cual sirve para ayudar al navegador a localizar el contenido en función de una ruta relativa al propio navegador en lugar de usar rutas físicas del propio equipo, las cuales podrían variar en función del Sistema Operativo.

5.3. Ficheros de contenido (*content*)

El directorio *content* contiene los ficheros XUL que definen la interfaz gráfica del usuario y los ficheros *JavaScript* que implementan toda la funcionalidad de la extensión. A continuación detallaremos todos los ficheros utilizados, explicando qué hace cada uno, y si es necesario, explicando las partes importantes del código incluido en los ficheros.

5.3.1. Interfaz de usuario

about.xul

Este fichero es la ventana de información de la propia extensión, y se utiliza para mostrar el nombre de la misma, la versión y el autor entre otros datos. En la figura 5.1 podemos ver cómo queda la ventana de información.

Figura 5.1: Ventana "Acerca de"

firefox-overlay.xul

Este fichero es el que se fusiona con el navegador de *Mozilla Firefox*, como ya se ha comentado en el archivo *chrome.manifest*, integrando en el menú de Herramientas del navegador un acceso a las opciones de la extensión. Además integra un acceso en el menú que se muestra cuando hacemos clic con el botón secundario. También referenciamos un fichero de propiedades con la etiqueta *</stringbundleset>*, para tener acceso a unas cadenas de texto con la finalidad de facilitar la traducción de la propia extensión sin introducir el texto directamente en el código. Este fichero de propiedades se llama *overlay.properties* y lo comentaremos en la sección 5.4. A continuación podemos ver cómo incluimos los menús de la extensión y los scripts que se necesitan dentro de la etiqueta *<overlay>*.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="chrome://folderTag/skin/overlay.css"
       type="text/css"?>
3
4  <!DOCTYPE overlay SYSTEM "chrome://folderTag/locale/overlay.
       dtd">
5
6  <overlay id="FolderTag-overlay" xmlns="http://www.mozilla.
       org/keymaster/gatekeeper/there.is.only.xul">
7
8    <script type="application/x-javascript" src="overlay.js"/>
9    <script type="application/x-javascript" src="firefox-
       overlay.js"/>
10   <script type="application/x-javascript" src="folderTag.js"
       />
11   <script type="application/x-javascript" src="sqlite.js"/>
12
```

```
13    <stringbundleset id="stringbundleset">
14      <stringbundle id="FolderTag-strings" src="chrome://
            folderTag/locale/overlay.properties"/>
15    </stringbundleset>
16
17    <menupopup id="menu_ToolsPopup">
18      <menuitem id="menu-FolderTag"
19             class="menuitem-iconic"
20             label="&folderTag.label;"
21             accesskey="&folderTag.accesskey;"
22             insertafter="devToolsSeparator"
23             oncommand="folderTag.onMenuItemCommand(event);"/>
24    </menupopup>
25
26    <popup id="contentAreaContextMenu">
27      <menu id="context-FolderTag"
28          class="menu-iconic"
29          label="&folderTagContext.label;"
30          accesskey="&folderTagContext.accesskey;"
31          insertafter="context-sep-stop">
32
33        <menupopup id="context-FolderTag-popup">
34          <menuitem id="opciones-FolderTag"
35                class="menuitem-iconic"
36                label="&folderTagPopup.opciones;"
37                accesskey="&folderTagPopup.opciones.accesskey;
                      "
38                oncommand="folderTag.onMenuItemCommand(event);
                      "/>
39
40          <menuitem id="actualizar-FolderTag"
41                class="menuitem-iconic"
42                label="&folderTagPopup.actualizar;"
43                accesskey="&folderTagPopup.actualizar.
                      accesskey;"
44                oncommand="folderTag.onMenuItemCommand(event);
                      "/>
45
46          <menuitem id="mostrar-FolderTag"
47                class="menuitem-iconic"
48                label="&folderTagPopup.mostrar;"
49                type="checkbox"
50                accesskey="&folderTagPopup.mostrar.accesskey;"
51                oncommand="folderTag.onMenuItemCommand(event);
                      "/>
52        </menupopup>
53      </menu>
54    </popup>
55  </overlay>
```

option.xul

Este fichero define la ventana principal de la extensión desde donde podremos organizar jerárquicamente nuestras etiquetas y donde almacenaremos nuestro identificador de usuario de *Delicious*. En la figura 5.2 podemos ver cómo sería la ventana de configuración.

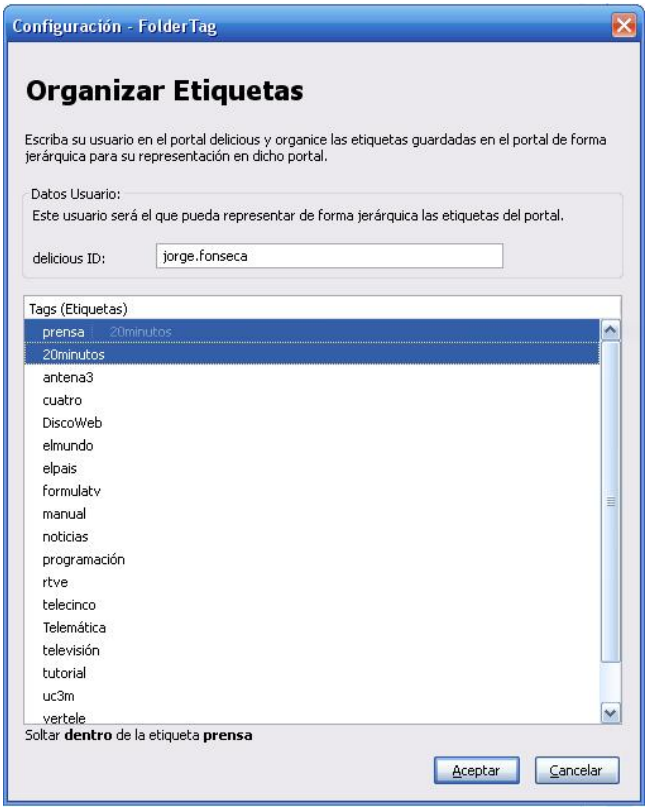

Figura 5.2: Ventana Configuración

5.3.2. Lógica de la extensión

firefox-overlay.js

Es el fichero desde el que se gestiona mediante eventos que cuando se pulse el botón secundario del ratón dentro del navegador, se muestre un acceso a las opciones de la extensión. También controla que solo se muestre si se está dentro del portal *Delicious*, ya que en caso contrario no mostrará el menú contextual que incluimos en el fichero *firefox-overlay.xul*. A continuación podemos ver cómo se agregan los eventos.

```
folderTag.onFirefoxLoad = function(event) {

  //Llama a la función encargada de mostrar el menuitem de la extensión cuando se pulsa el botón secundario del ratón
  document.getElementById("contentAreaContextMenu")
      .addEventListener("popupshowing", function (e){ folderTag.showFirefoxContextMenu(e); }, false);
};

folderTag.showFirefoxContextMenu = function(event) {

  // Muestra u oculta el menuitem en base al menú contextual en el que está y si está dentro de la página delicious
  if(window.content.document.location.href.indexOf('delicious.com/') != -1){
    document.getElementById("context-FolderTag").hidden = gContextMenu.onImage;//Si el foco no está en una imagen se mostrará el menuitem
  }else{
    document.getElementById("context-FolderTag").hidden = true;
  }

};

window.addEventListener("load", folderTag.onFirefoxLoad, false);
```

folderTag.js

Este fichero agrega un evento al navegador cuando se carga el contenido DOM de una página dentro del navegador para crear la base de datos de la extensión si no se hubiera creado ya. Es aquí donde creamos el archivo que contendrá la base de datos y donde creamos las tablas y los *triggers* de borrado en cascada para mantener la integridad referencial de la base de datos. También controla que se muestre el menú jerárquico de la extensión dentro de la página de *Delicious* si el usuario ha marcado dicha opción. A continuación podemos ver cómo creamos la base de datos.

```
//Obtenemos el objeto nsIFile del archivo, si no existiera no crearía el archivo pero si la ruta de donde estaría
Components.utils.import("resource://gre/modules/FileUtils.jsm");
var file = FileUtils.getFile("ProfD", ["folderTag", "folderTag.sqlite"]);

```

```
5  //Comprobamos si ya está creada la base de datos
6  if(!file.exists()) {
7    //Creamos la base de datos y las tablas
8    var bbdd = new SQLite("folderTag.sqlite", {location:'ProfD
         '});
9    bbdd.execute("CREATE TABLE IF NOT EXISTS Usuario (id TEXT
         PRIMARY KEY)");
10   bbdd.execute("CREATE TABLE IF NOT EXISTS Tag (etiqueta
         TEXT PRIMARY KEY)");
11   bbdd.execute("CREATE TABLE IF NOT EXISTS Relacion (padre
         TEXT, hijo TEXT, orden INTEGER NOT NULL, PRIMARY KEY(
         hijo), FOREIGN KEY (padre) references Tag(etiqueta) ON
         DELETE CASCADE, FOREIGN KEY (hijo) references Tag(
         etiqueta) ON DELETE CASCADE)");
12   bbdd.execute("CREATE TABLE IF NOT EXISTS Raiz (padre TEXT
         PRIMARY KEY, orden INTEGER NOT NULL, FOREIGN KEY (padre
         ) references Tag(etiqueta) ON DELETE CASCADE)");
13   bbdd.execute("PRAGMA foreign_keys = ON;");
14   bbdd.execute("CREATE TRIGGER IF NOT EXISTS
         Insertar_Relacion_Padre BEFORE INSERT ON Relacion FOR
         EACH ROW BEGIN SELECT RAISE (ROLLBACK, 'No se puede
         inserta el registro porque no existe la etiqueta padre
         ') WHERE (SELECT etiqueta FROM Tag WHERE etiqueta = NEW
         .padre) IS NULL; END");
15   bbdd.execute("CREATE TRIGGER IF NOT EXISTS
         Insertar_Relacion_Hijo BEFORE INSERT ON Relacion FOR
         EACH ROW BEGIN SELECT RAISE (ROLLBACK, 'No se puede
         inserta el registro porque no existe la etiqueta hijo')
          WHERE (SELECT etiqueta FROM Tag WHERE etiqueta = NEW.
         hijo) IS NULL; END");
16   bbdd.execute("CREATE TRIGGER IF NOT EXISTS
         Insertar_Raiz_Hijo BEFORE INSERT ON Relacion FOR EACH
         ROW BEGIN SELECT RAISE (ROLLBACK, 'No se puede inserta
         el registro porque una etiqueta raíz no puede ser hijo
         ') WHERE (SELECT padre FROM Raiz WHERE padre = NEW.hijo
         ) IS NOT NULL; END");
17   bbdd.execute("CREATE TRIGGER IF NOT EXISTS Insertar_Raiz
         BEFORE INSERT ON Raiz FOR EACH ROW BEGIN SELECT RAISE (
         ROLLBACK, 'No se puede inserta el registro porque no
         existe la etiqueta padre') WHERE (SELECT etiqueta FROM
         Tag WHERE etiqueta = NEW.padre) IS NULL; END");
18   bbdd.execute("CREATE TRIGGER IF NOT EXISTS Borrar_Relacion
          BEFORE DELETE ON Tag FOR EACH ROW BEGIN DELETE FROM
         Relacion WHERE padre = OLD.etiqueta OR hijo = OLD.
         etiqueta; END");
19   bbdd.execute("CREATE TRIGGER IF NOT EXISTS Borrar_Raiz
         BEFORE DELETE ON Tag FOR EACH ROW BEGIN DELETE FROM
         Raiz WHERE padre = OLD.etiqueta; END");
20 }
```

folderTagTreeView.js

Este fichero contiene toda la funcionalidad para representar las etiquetas de forma jerárquica mediante una estructura de árbol. Incluye funciones para mostrar u ocultar los hijos de un nodo del árbol, toda la funcionalidad para la función de arrastrar y soltar y la función para almacenar la estructura del árbol del menú de configuración. Para el manejo de la estructura árbol se trabaja con dos matrices: *childData* que contiene todas las relaciones entre etiquetas [padre, hijo-0, hijo-1, hijo-2, ..., hijo-n], y *visibleData* que contiene todas las etiquetas visibles en el momento [tag, isContainerEmpty, nivel, isContainerOpen]. Las funciones de la interfaz *nsITreeView* las definimos para trabajar sobre estas matrices y así poder representar la estructura jerárquica y poder controlar los eventos de *drag & drop*.

option.js

Este fichero es el encargado de mostrar en la ventana de configuración el usuario y las etiquetas que haya almacenado en la propia base de datos. También se encarga de guardar los cambios que se produzcan tanto en el usuario como la estructura del árbol de etiquetas. La representación de la estructura jerárquica de las etiquetas se realiza con las funciones creadas en el fichero *folderTagTreeView.js*.

overlay.js

Este fichero contiene toda la funcionalidad del menú de opciones de la extensión. Detecta mediante eventos qué botón se ha pulsado para mostrar la ventana de configuración, actualizar las etiquetas que tiene el usuario almacenadas en su perfil de *Delicious* o mostrar/ocultar la estructura jerárquica de las etiquetas en el propio portal de *Delicious*. Cuando se selecciona mostrar la ventana de configuración abre un diálogo invocando la ventana creada en el fichero *option.xul*. Para actualizar las etiquetas primero comprobamos que se esté identificado en el portal con el usuario definido, y después se comprueba etiqueta a etiqueta si ya estaba almacenada para insertarla o continuar con la siguiente. Una vez terminado de insertar todas las etiquetas se eliminarían las que ya no existieran. Para mostrar la estructura jerárquica vamos comprobando las tablas y creando los nodos HTML siguiendo la estructura almacenada. Para ocultar dicha estructura simplemente ocultamos el nodo raíz que contiene todos los elementos mediante estilos CSS.

sqlite.js

Este fichero es el que gestiona la base de datos SQlite de la extensión. Se encarga tanto de crear un fichero que contendrá la base de datos, como de ejecutar las sentencias SQL que se pasen.

5.4. Ficheros de configuración regional (*locale*)

En este directorio se encuentran los ficheros DTD y los ficheros *properties* de la extensión. Los ficheros DTD son utilizados para las traducciones de los ficheros XUL, mientras que los ficheros *properties* son usados para las traducciones de las cadenas en el código *JavaScript*. Todos estos ficheros están dentro del subdirectorio *es-ES* porque nuestra extensión está destinada para los usuarios de habla castellano. Si quisiéramos traducir la extensión a otros idiomas bastaría con crear el subdirectorio pertinente que contenga los mismos ficheros con los textos traducidos. Cuando se referencian los ficheros de configuración regional no se incluye el idioma a utilizar en la URL, ya que el registro chrome resuelve las URIs basándose en la configuración regional actual y la información proporcionada en el archivo *chrome.manifest*. A continuación veremos un pequeño extracto de ambos ficheros y cómo referenciarlos.

Como ejemplo de un fichero DTD tenemos el contenido de *overlay.dtd*, donde se puede ver que mediante los elementos *ENTITY* relacionamos el nombre que se utiliza en el fichero XUL para referenciar el texto a traducir con el mensaje traducido. No solamente se utilizan los DTD para mostrar texto traducido; también se puede ver cómo en las líneas pares declaramos los atajos de teclado de los botones que traducimos, ya que intentamos usar la primera letra del botón y esto dependiendo de la traducción podría cambiar.

```
1  <!ENTITY folderTag.label "FolderTag">
2  <!ENTITY folderTag.accesskey "F">
3  <!ENTITY folderTagContext.label "FolderTag">
4  <!ENTITY folderTagContext.accesskey "F">
5  <!ENTITY folderTagPopup.opciones "Configuración">
6  <!ENTITY folderTagPopup.opciones.accesskey "C">
7  <!ENTITY folderTagPopup.actualizar "Actualizar Etiquetas">
8  <!ENTITY folderTagPopup.actualizar.accesskey "A">
9  <!ENTITY folderTagPopup.mostrar "Mostrar/Ocultar FolderTag">
10 <!ENTITY folderTagPopup.mostrar.accesskey "M">
```

Luego en el fichero XUL la referencia del texto se indica como podemos ver en las líneas 5 y 6 del siguiente extracto del fichero *firefox-overlay.xul*. Además, en la línea 1 del extracto, podemos ver cómo referenciamos el fichero DTD dentro del fichero XUL para que utilice las traducciones.

```
1  <!DOCTYPE overlay SYSTEM "chrome://folderTag/locale/overlay.
       dtd">
2  ...
3  <menuitem id="actualizar-FolderTag"
4    class="menuitem-iconic"
5    label="&folderTagPopup.actualizar;"
6    accesskey="&folderTagPopup.actualizar.accesskey;"
7    oncommand="folderTag.onMenuItemCommand(event);"/>
```

Como ejemplo de un fichero *properties* tenemos el contenido de *overlay.properties*, donde se puede ver que relaciona una variable con su correspondiente texto traducido mediante el símbolo =. Este fichero luego es referido en el fichero XUL mediante la etiqueta *</stringbundleset>*, para poder obtener mediante las funciones *getElementById(idEtiqueta)* acceso al fichero *properties*, y *getString(variable)* acceso al texto de las cadenas.

```
usuario=No hay un usuario definido. Por favor, introduzca un
    usuario en la extensión FolderTag.
usuarioBorrado=Se ha eliminado el usuario almacenado debido
    a algún error. Por favor, vuelva a introducirlo.
login=No está identificado en el portal delicious. Por favor
    , identifíquese en el portal delicious.
urlEtiquetas=No está en la página donde aparecen todas las
    etiquetas del usuario definido. Diríjase al listado de
    todas las etiquetas de su perfil para poder actualizarlas
    .
actualizar=A continuación se actualizarán las etiquetas de
    su perfil en el portal delicious. Recuerde darle a
    mostrar todas las etiquetas en el caso de tener muchas y
    no mostrarse todas.\n\nEsta acción puede tardar varios
    segundos dependiendo de la cantidad de etiquetas
    existentes. ¿Desea actualizar las etiquetas?
error=Ha ocurrido un error inesperado y no se han podido
    actualizar las etiquetas. Asegúrese de estar en el
    listado de las etiquetas de su perfil y que aparezcan
    todas.
```

Aquí podemos ver un extracto del fichero *firefox-overlay.xul* para ver cómo se referencia el fichero dentro del archivo XUL:

```
...
<stringbundleset id="stringbundleset">
  <stringbundle id="FolderTag-strings" src="chrome://
      folderTag/locale/overlay.properties"/>
</stringbundleset>
...
```

A continuación podemos ver cómo obtener acceso al valor de una cadena de texto dentro del archivo *JavaScript*:

```
...
texto = document.getElementById("FolderTag-strings").
    getString('usuarioBorrado');
...
```

5.5. Ficheros de aspecto (*skin*)

En este directorio se encuentran todos los ficheros CSS y las posibles imágenes que use nuestra extensión contenidas en un subdirectorio denominado *images*. Los ficheros CSS han sido usados para definir la apariencia de la interfaz y así poder tener controlado el estilo de la interfaz separado de la estructura de los documentos XUL. Las imágenes que se han utilizado han sido creadas por nosotros y obtenidas de alguna web como *OpenClipArt* donde las imágenes no tienen ningún tipo de derecho para su uso libre. A continuación veremos un pequeño extracto de un fichero CSS para ver cómo configuramos la apariencia de las ventanas XUL:

```
1  #menu-FolderTag
2  {
3    list-style-image: url("chrome://folderTag/skin/images/
         folderTag16.png");
4  }
5
6  #context-FolderTag
7  {
8    list-style-image: url("chrome://folderTag/skin/images/
         folderTag16.png");
9  }
10
11 #FolderTag-status-image
12 {
13   list-style-image: url("chrome://folderTag/skin/images/
         folderTag16.png");
14 }
15
16 #opciones-FolderTag
17 {
18   list-style-image: url("chrome://folderTag/skin/images/
         menu_icon.png");
19 }
20
21 #actualizar-FolderTag
22 {
23   list-style-image: url("chrome://folderTag/skin/images/
         refresh.png");
24 }
```

Este es el contenido del fichero *overlay.css*, donde podemos ver que añadimos las imágenes que tenemos almacenadas en los botones del menú de nuestra extensión. En este ejemplo sólo añadimos imágenes al lado del botón, pero podríamos modificar los márgenes, la fuente y los colores, entre otros elementos del archivo XUL.

5.6. Consideraciones generales

Durante el desarrollo de esta extensión hemos sufrido el cambio de versiones tanto en el navegador *Mozilla Firefox* como en el portal *Delicious*. En lo referente al cambio de versiones del navegador no nos afectó demasiado. Quizás hizo que que las opciones de la extensión integradas en el menú de Herramientas del navegador quedaran un poco obsoletas puesto que a partir de la versión 4.0 desaparecía la barra de menús del navegador. Pero al disponer de un menú integrado en el menú contextual, cuando se hace clic con el botón secundario, no nos hizo tener que hacer cambios. Es por ello que nuestra extensión funciona desde la versión 3.6, con la que comenzamos a desarrollarla, hasta la versión actual en el momento de escribir esta memoria que es la 41.

Los cambios de versiones del portal *Delicious* sí tuvieron un mayor impacto en el desarrollo de la extensión. Los cambios que más nos afectaron fueron los que hicieron en el diseño del portal. Por un lado cambiaron en varias ocasiones el estilo visual del portal, lo que nos hizo ir adaptando el estilo del menú jerárquico que incrustamos en el portal para que fuera acorde. Podemos ver en la figura 5.3 uno de los estilos anteriores comparado con el actual (la imagen de arriba es de 2013 y la de abajo de la actualidad). Otro de los cambios fue el listado de etiquetas, que en versiones anteriores había que ir a una página en concreto para visualizar todas, mientras que en la versión actual se encuentran ocultas por defecto en el portal aunque el usuario las puede hacer visibles en cualquier momento. Y por último, unos cambios internos en la codificación del HTML de la página nos hicieron rehacer las consultas XPath para controlar si el usuario estaba autenticado en el portal y con qué identificador de usuario.

En la implementación de la extensión también tuvimos que solucionar algunos inconvenientes. En lo referente al uso de la base de datos SQlite, nos encontramos que por defecto las claves externas no son soportadas, por lo que a la hora de borrar en cascada cuando eliminamos una etiqueta se hizo necesario el uso de *TRIGGERS*. Otro problema a solucionar que nos encontramos a la hora de desarrollar la extensión fueron los tiempos de acceso a la base de datos. Al principio, cuando íbamos actualizando las etiquetas, íbamos accediendo a la base de datos por cada inserción que hacíamos, lo que nos podía suponer unos tiempos superiores al minuto en la actualización de las etiquetas. Este problema lo solucionamos accediendo una única vez a la base de datos con las sentencias *BEGIN IMMEDIATE TRANSACTION* y *COMMIT TRANSACTION*, metiendo entre medias todos los *INSERT* que hubiera, consiguiendo reducir los tiempos de actualización a cuestión de segundos.

Finalmente, como consideraciones generales y requisitos para el funcionamiento de la extensión, decidimos que es imprescindible estar registrado en el el portal *Delicious* y disponer de etiquetas almacenadas, ya que la extensión está diseñada para ayudar en la organización de las mismas y no para crear nuevas. También que fuera necesario estar identificado en el portal ya que el usuario podría tener etiquetas privadas que no sean visibles al resto de usuarios. Además, el acceso al menú de configuración debía ser accesible en cualquier momento porque no es necesario encontrarse en el portal, pero el resto de funcio-

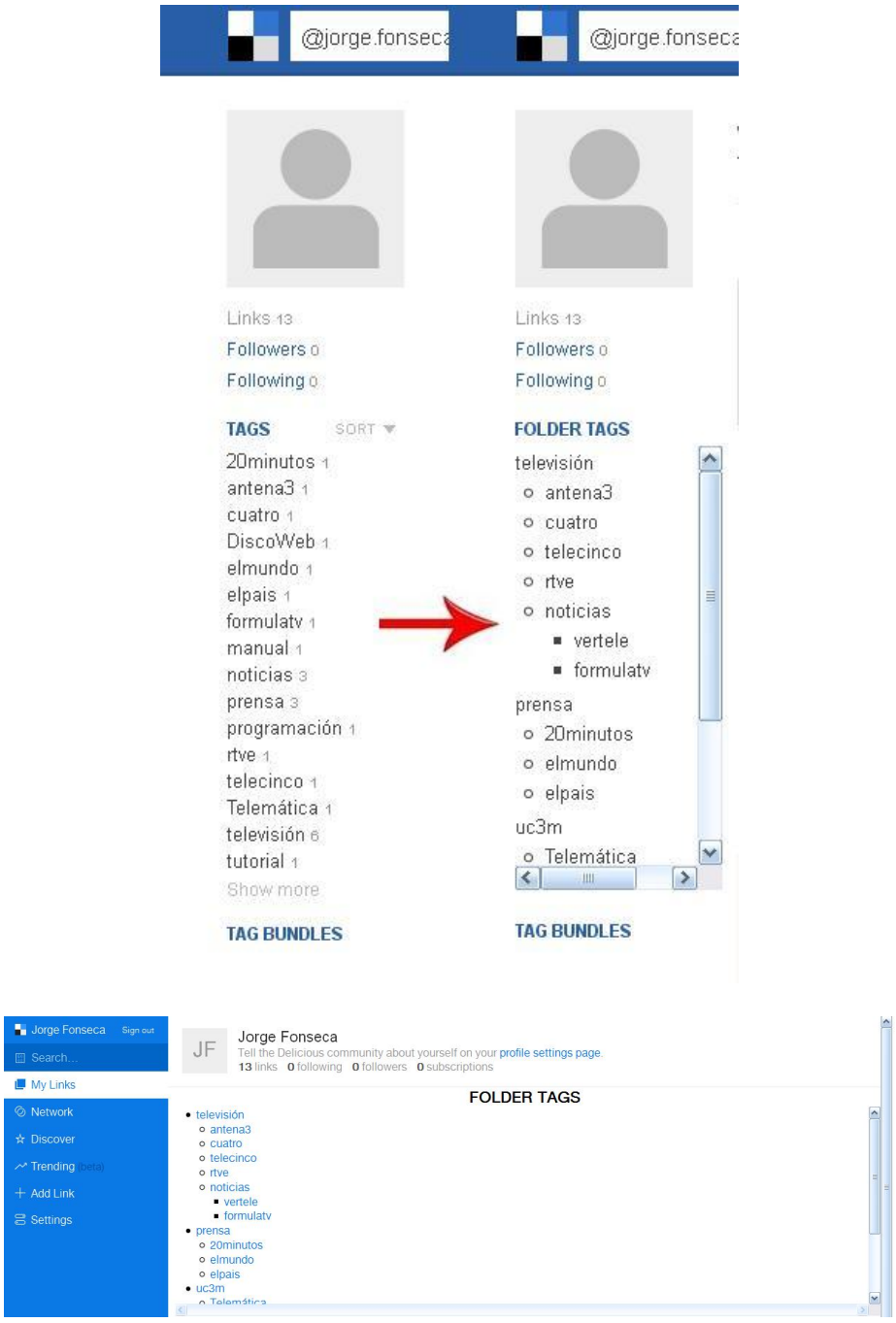

Figura 5.3: Cambio de versión en el portal Delicious.

nalidades solamente estarían disponibles cuando se estuviera dentro del dominio *Delicious,* pulsando el botón secundario del ratón, ya que dichas funcionalidades interactúan con el portal. Finalmente, el almacenamiento o actualización de las etiquetas no se obtendría de forma automática puesto que sería necesario pedir la contraseña del usuario y por motivos de seguridad y confianza descartamos esa opción.

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo se presentarán las conclusiones a las que hemos podido llegar tras la realización del proyecto y se discutirá si se han alcanzado los objetivos propuestos al inicio del mismo. También se presentarán los posibles trabajos futuros a desarrollar que se podrían sacar a raíz de este proyecto.

6.1. Conclusiones

Durante el desarrollo del proyecto hemos podido sacar una serie de conclusiones en diferentes ámbitos tales como en el estudio del modelo de clasificación por etiquetado jerárquico como en el desarrollo de extensiones para el navegador *Mozilla Firefox*.

En primer lugar hemos podido ver la importancia que tiene en nuestros días la información así como el manejo y tratamiento de la misma. Actualmente la información proporcionada por un programa informático es tratada según la relevancia que tenga con los parámetros dados por el usuario, devolviendo un bloque de datos donde se puede llegar a incluir información no útil o irrelevante. Y esta situación es la que se quiere cambiar, de forma que la comunicación con los programas informáticos sea mucho más natural, siendo éstos capaces de entender lo que se solicita y dar la información concreta y precisa excluyendo el resto de datos. De acuerdo con este propósito hemos querido relacionar un sistema de etiquetado con uno jerárquico para establecer relaciones entre los términos y así poder obtener resultados más precisos en las búsquedas.

Por otro lado hemos podido explorar y profundizar en el ámbito de las extensiones para el navegador *Mozilla Firefox*. Se ha comprobado que el desarrollo de estas aplicaciones puede llegar a ser muy potente gracias al lenguaje XUL para las interfaces gráficas unido a otros lenguajes, como son *JavaScript* y CSS, y a la posibilidad de usar bases de datos, como puede ser SQLITE. De esta manera es posible desarrollar múltiples diseños y funcionalidades que son añadidas

al propio navegador, e incluso pequeñas aplicaciones independientes integradas dentro del navegador.

Dicho todo esto, podemos concluir que los objetivos propuestos al inicio del desarrollo del proyecto han sido cumplidos. Se ha logrado desarrollar una extensión para un navegador que permite representar un modelo de clasificación por etiquetado jerárquico en un aplicación web. Además se ha seguido la arquitectura "Modelo Vista Controlador" para permitir su fácil adaptabilidad a posibles cambios ajenos como ha sucedido a lo largo de la realización del proyecto debido a los cambios de diseño del portal *Delicious*. Y hemos realizado un estudio de los modelos de clasificación más utilizados en aplicaciones web como son el etiquetado y el jerárquico, junto con las tecnologías empleadas para el desarrollo de la extensión.

6.2. Trabajos futuros

Como se ha podido ver a lo largo de esta memoria, este proyecto está muy centrado para un caso concreto pero tiene muchas posibilidades de expansión y mejora. Por este motivo, a continuación comentaremos las posibles líneas futuras a desarrollar del mismo.

La principal línea de desarrollo futura que se puede plantear es escalar la extensión a múltiples portales web. Como ya hemos dicho, este proyecto está centrado para el caso del portal *Delicious*, pero se podría ampliar su funcionalidad para otros donde se utilice un sistema de etiquetado para catalogar la información. Este desarrollo podría realizarse fácilmente solamente analizando dónde tiene el listado de etiquetas la aplicación en la que se quiera implantar ya que debido a la arquitectura de "Modelo Vista Controlador" el resto de la lógica ya está desarrollada teniendo que modificar sólo el modelo. También sería necesario realizar alguna pequeña modificación en el diseño de la base de datos para soportar la funcionalidad en múltiples aplicaciones web.

Otra posible línea futura sería desarrollar esta extensión para múltiples navegadores evitando estar limitada a un único navegador, ya que los posibles usuarios de la herramienta puede que usen otros navegadores diferentes al escogido para el desarrollo de la extensión y estaríamos forzándoles a cambiar o usar ambos si están interesados en la utilización de la extensión.

Como posibles trabajos de mejora secundarios también tenemos la posibilidad de soportar múltiples usuarios para una misma aplicación, ya que descartamos esta opción en el desarrollo inicial porque consideramos que la extensión es usada por una única persona por equipo y si compartiera el equipo usarían diferentes perfiles. También se podría mejorar el diseño HTML a la hora de incrustar el método de clasificación por etiquetado jerárquico porque simplemente intentamos respetar el propio estilo del portal sin personalizar el nuevo menú. Un ejemplo de esto sería que según se seleccione una etiqueta padre del menú jerárquico ésta se despliegue o contraiga.

Además de todas estas mejoras futuras, existe otra vertiente de desarrollo dejando un poco al margen el sistema de clasificación por etiquetado jerárquico.

Como ya hemos comentado al inicio de la memoria, nuestra motivación a la hora de desarrollar este proyecto partía de la tendencia en la actualidad a intentar tratar mejor los datos y sacar información de ellos. Es por ello, que con los debidos cambios, esta extensión podría ser utilizada para recoger información o datos de la propia navegación de un usuario. Ahora mismo, la extensión recoge un listado de etiquetas del portal *Delicious* para permitir su relación, pero se podría modificar para que según unas palabras clave introducidas pudiera ir detectándolas durante la navegación del usuario y recopilando información y las URL de donde las ha encontrado. Este desarrollo se saldría un poco de la línea seguida hasta ahora donde nos centrábamos en los modelos de clasificación y supondría realizar un nuevo estudio y análisis, pero no deja de ser una posible línea futura a explorar dentro de la gestión de la información en aplicaciones web.

Capítulo 7

Presupuesto

En este capítulo se realizará un análisis del desarrollo de este proyecto en términos económicos. Se describirán los recursos utilizados, se detallarán las tareas realizadas divididas en diferentes fases y se realizará una estimación económica de los costes producidos.

7.1. Recursos

A continuación describiremos todos los recursos empleados durante el desarrollo del proyecto, los cuales han sido divididos en tres tipos: personal, hardware/software y fungibles.

7.1.1. Personal

- *Luis de la Fuente Valentín*, Ingeniero de Telecomunicaciones por la Universidad de Valladolid y Doctor en Ingeniería Telemática por la Universidad Carlos III de Madrid como tutor de este proyecto durante las partes de diseño y desarrollo.
- *Carlos Alario Hoyos*, Ingeniero de Telecomunicaciones por la Universidad de Valladolid y Doctor en Tecnologías de la Información y las Comunicaciones por la Universidad de Valladolid como tutor de este proyecto en las etapas finales.
- *Jorge Fonseca Mesonero*, alumno de la titulación Ingeniería Técnica de Telecomunicaciones en la especialidad de Telemática en la Universidad Carlos III de Madrid como autor de este proyecto siguiendo las directrices definidas por los tutores.

7.1.2. Hardware/Software

- Ordenador PC *HP* con procesador Intel Core 2 Duo y 2 GB de memoria RAM.

- Sistema Operativo Windows XP SP3.
- Navegador *Mozilla Firefox*.
- Extensiones Firebug y SQLite Manager.
- Editor LyX.
- Notepad ++.

7.1.3. Fungibles

- Electricidad.
- Material de oficina.
- Conexión a Internet.

7.2. Planificación

En la tabla 7.1 vemos las distintas fases del proyecto con una breve descripción de las tareas realizadas, y a continuación, en la figura 7.1 observaremos el diagrama de Gantt correspondiente.

Fase	Descripción	Duración (días/horas)
Estudio previo	Recogida de información inicial sobre el contexto del proyecto	7/56
Adquisición de conocimiento	Consulta de documentación sobre las tecnologías usadas y realización de una extensión "Hola Mundo"	13/104
Análisis	Estudio de los problemas de la clasificación por etiquetado y la clasificación jerárquica y selección de la clasificación por etiquetado jerárquico	10/80
Diseño	Definición de requisitos, diseño de la extensión (base de datos, interfaz gráfica y funcionalidad) y entorno de aplicación	20/160
Implementación	Implementación de la extensión y comprobación de funcionamiento	70/560
Actualización	Correcciones en la extensión para mantener su funcionamiento debido a cambios en las versiones del navegador y actualizaciones de la aplicación web *Delicious*	15/120
Documentación	Realización de la memoria del proyecto	45/360
	Total	180/1440

Tabla 7.1: Planificación de tareas

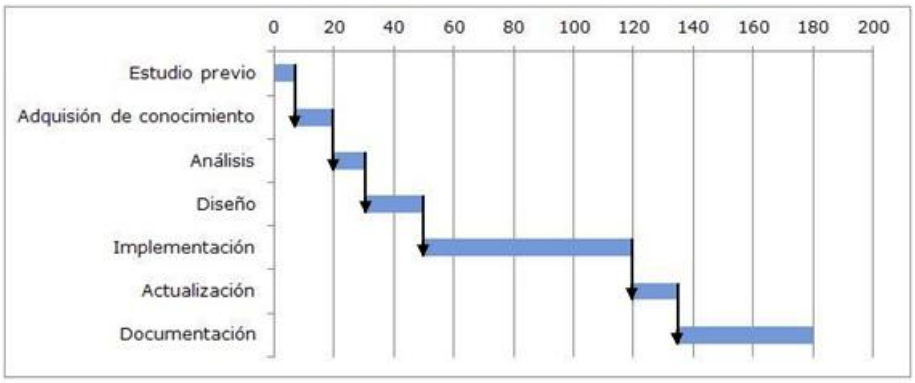

Figura 7.1: Diagrama Gantt

7.3. Costes

A continuación detallaremos en sucesivas tablas los costes relacionados con los distintos recursos utilizados.

7.3.1. Personal

En el cálculo del coste del personal incluiremos tanto el coste del alumno que ha realizado el proyecto como el coste de los tutores, estimando que éstos últimos han dedicado un 10 % del tiempo total del proyecto. El precio de la jornada[1] se ha estimado en 115€ para un analista programador junior y en 180€ para un analista programador senior basándonos en una ponderación del mercado laboral[2]. En la tabla 7.2 vemos el coste del personal.

Trabajador	Jornadas	Precio jornada	Coste
Analista programador junior	180 días	120 €	21.600 €
Analista programador senior	18 días	180 €	3.240 €

Tabla 7.2: Coste personal

7.3.2. Material

Al tratarse de software libre no se han requerido licencias de uso por lo que sólo se ha tenido en cuenta el procedente del equipo. Se ha calculado el coste de amortización del equipo siguiendo la fórmula de amortización siguiente:

$$\frac{A}{B} \times C \times D$$

[1] Se considera que una jornada son 8 horas trabajadas en un día.

[2] Se ha tenido en cuenta unos sueldos brutos anuales de 20.000€ para el programador junior y 30.000€ para el programador senior, un 30 % las cotizaciones de la empresa para dichos sueldos y un año laboral de 220 días. Dichos cálculos se han redondeado al alza.

Donde A es el número de meses desde la fecha de facturación que el equipo es utilizado[3], B es el periodo de depreciación, C el coste del equipo sin IVA y D el porcentaje de uso que se dedica al proyecto. En la tabla 7.3 vemos el coste material.

Descripción	A	B	C	D	Coste imputable
PC de sobremesa y S.O. Windows XP	9 meses	60 meses	700 €	100 %	105 €

Tabla 7.3: Coste material

7.3.3. Indirectos

Los costes indirectos del proyecto son los que derivan de los costes de funcionamiento que no se pueden asignar únicamente a la realización del proyecto. Se hace una estimación del 20 % de los costes directos. En la tabla 7.4 vemos los costes indirectos.

Descripción	Costes
Electricidad, material, conexión a Internet...	4.989€

Tabla 7.4: Costes indirectos

7.3.4. Total

El presupuesto total de este proyecto, teniendo en cuenta los costes mencionados anteriormente, asciende a la cantidad de **29.934 EUROS**.

Leganés a 21 de Septiembre de 2015

El ingeniero proyectista

Fdo: Jorge Fonseca Mesonero

[3] 180 días entre 20 días laborables por mes

Glosario

ANSI	American National Standards Institute
API	Application Programming Interface
BLOB	Binary Large Objects
BSON	Binary JSON
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DBMS	DataBase Management System
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
DOM	Document Object Model
GUID	Globally Unique Identifier
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
RDBMS	Relational DataBase Management System
RSS	Really Simple Syndication
SQL	Structured Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XPath	XML Path Language
XPCOM	Cross-platform Component Object Model
XUL	XML-based User-interface Language

Bibliografía

[1] ¿Cuánta Información se Genera y Almacena en el Mundo?, <https://documania20.wordpress.com/2013/09/16/cuanta-informacion-se-genera-y-almacena-en-el-mundo/>, Accedido en Septiembre de 2015

[2] Introducción al concepto Big Data, <http://www.cnis.es/index.php?option=com_content&view=article&id=434:concepto-big-data&catid=47:noticias-boletin-cnis&Itemid=57>, Accedido en Septiembre de 2015

[3] La moda del Big Data: ¿En qué consiste en realidad?, <http://www.eleconomista.es/tecnologia/noticias/5578707/02/14/La-moda-del-Big-Data-En-que-consiste-en-realidad.html#.Kku8doHWJdEAqJ9>, Accedido en Septiembre de 2015

[4] Descripción Web Semántica, <http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica>, Accedido en Septiembre de 2015

[5] Flor Nancy Díaz Piraquive, Luis Joyanes Aguilar, Víctor Hugo Medina García: "Taxonomía, ontología y folksonomía, ¿qué son y qué beneficios u oportunidades presentan para los usuarios de la web?", Universidad y Empresa, vol. 8, n.º 16, pp. 242-261 (Enero-Junio, 2009)

[6] Qué es MVC, <http://www.desarrolloweb.com/articulos/que-es-mvc.html>, Accedido en Septiembre de 2015

[7] Apuntes de la asignatura "Análisis de la sociedad de la información", 2008-2009

[8] Minería de Datos, <http://www.uoc.edu/web/esp/art/uoc/molina1102/molina1102.html>, Accedido en Septiembre de 2015

[9] Sistemas de clasificación de información, <http://www.nosolousabilidad.com/articulos/sistemas_clasificacion.htm>, Accedido en Septiembre de 2015

[10] Folksonomías, <http://www.infotecarios.com/de-las-folksonomias-o-la-torre-de-babel/>, Accedido en Septiembre de 2015

[11] Silvia Argudo, Miquel Centelles: "Metodología para el diseño de taxonomías corporativas", Investigación Bibliotecológica, vol. 19, n.º 39, pp. 158-177 (Julio-Diciembre, 2005)

[12] Evolución de la Web, <https://sietepecadosdigitales.wordpress.com/2012/02/27/evolucion-de-la-web/>, Accedido en Septiembre de 2015

[13] La tendencia de la Web, <http://manuelgross.bligoo.com/content/view/588406/La-tendencia-de-la-Web-Desde-la-1-0-hasta-la-5-0.html>, Accedido en Septiembre de 2015

[14] Características de Mozilla Firefox, <https://www.mozilla.org/es-ES/firefox/desktop/>, Accedido en Septiembre de 2015

[15] Creando una extensión, <https://developer.mozilla.org/en-US/docs/Building_an_Extension>, Accedido en Septiembre de 2015

[16] Interfaces XPCOM, <https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/XPCOM_Interfaces>, Accedido en Septiembre de 2015

[17] Drag & Drop, <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Drag_and_drop>, Accedido en Septiembre de 2015

[18] Interfaz Tree View, <https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsITreeView>, Accedido en Septiembre de 2015

[19] Documentación Tree View básico, <https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/Custom_Tree_Views>, Accedido en Septiembre de 2015

[20] Documentación Tree View jerárquico, <https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/Tree_View_Details>, Accedido en Septiembre de 2015

[21] Tutorial JavaScript, <http://www.w3schools.com/js/js_intro.asp>, Accedido en Septiembre de 2015

[22] Introducción a JavaScript, <http://librosweb.es/libro/javascript/>, Accedido en Septiembre de 2015

[23] Tutorial XPath, <http://www.w3schools.com/xpath/default.asp>, Accedido en Septiembre de 2015

[24] Introducción a XPath en JavaScript, <https://developer.mozilla.org/en-US/docs/Introduction_to_using_XPath_in_JavaScript>, Accedido en Septiembre de 2015

[25] Asignación de DOM para XPath, <http://www.w3.org/TR/DOM-Level-3-XPath/xpath.html>, Accedido en Septiembre de 2015

[26] Tutorial SQL, <http://www.w3schools.com/sql/default.asp>, Accedido en Septiembre de 2015

[27] Manual SQL, <http://www.desarrolloweb.com/manuales/9/>, Accedido en Septiembre de 2015

[28] Apuntes de la asignatura "Bases de Datos Distribuidas", 2007-2008

[29] Tutorial de XUL, <https://developer.mozilla.org/es/docs/Tutorial_de_XUL>, Accedido en Septiembre de 2015

[30] Características de SQLite, <http://www.empresayeconomia.es/aplicaciones-para-empresas/sqlite-el-motor-de-base-de-datos-agil-y-robusto.html>, Accedido en Septiembre de 2015

[31] SQLite, <http://www.sqlite.org/index.html>, Accedido en Septiembre de 2015

[32] API SQLite Mozilla Firefox, <https://developer.mozilla.org/en-US/docs/Storage>, Accedido en Septiembre de 2015

Apéndice A

Manuales de la extensión FolderTag

En este anexo se describirá cómo instalar la extensión en el navegador *Mozilla Firefox* y el funcionamiento de la extensión para facilitar su utilización por parte del usuario. Esta extensión ha sido probada desde la versión 3.6 hasta la actual del navegador. Dado que está orientada al portal *Delicious* se presupone que se tiene una cuenta en dicho portal con etiquetas creadas, ya que esta extensión solamente ofrece la posibilidad de organizar y no de crear etiquetas nuevas.

A.1. Manual de instalación

Para la instalación de esta extensión necesitaremos el archivo instalable con extensión xpi tal y como comentamos en la sección 3.1.3. En nuestro caso, el fichero se llamará *folderTag.xpi* y será necesario abrirlo con el navegador *Mozilla Firefox* para su instalación.

Esto se puede hacer de varias maneras:

1. Abriendo el archivo a través del propio navegador mediante el menú **Firefox** > **Nueva Pestaña** > **Abrir archivo** y seleccionando el fichero *folderTag.xpi*. Ver figura A.1.

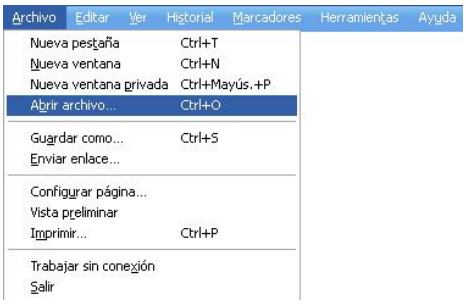

Figura A.1: Método 1 de instalación

2. Seleccionando directamente con el botón secundario del ratón el archivo y eligiendo la opción de **Abrir con** > **Firefox**. Si el archivo ya tuviera asociada la opción por defecto de abrir con *Firefox*, sólo sería necesario hacer doble clic para su instalación. Ver figura A.2.

Figura A.2: Método 2 de instalación

3. Arrastrando el archivo directamente al navegador *Mozilla Firefox* o al acceso directo del mismo. Ver figura A.3.

Figura A.3: Método 3 de instalación

Una vez abierto el archivo con el navegador *Mozilla Firefox*, nos aparecerá la ventana de **Instalación de software** en dicho navegador. Esta ventana informa al usuario sobre el complemento que va a instalar, indicando el nombre, la ruta del archivo y el autor si estuviera verificado[1]. Para continuar con la instalación pulsaremos el botón **Instalar ahora**. Ver figura A.4.

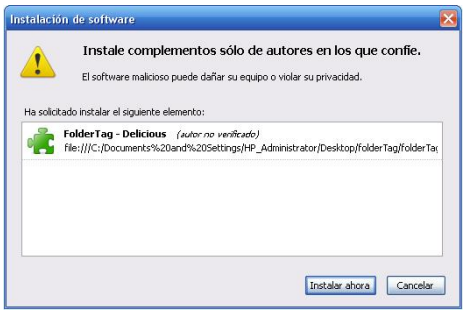

Figura A.4: Ventana de instalación de software

Cuando se haya completado la instalación de la extensión se abrirá una ventana indicando que para realizar los cambios será necesario reiniciar el navegador. Pulsaremos el botón **Reiniciar Firefox** para finalizar la instalación. Ver figura A.5.

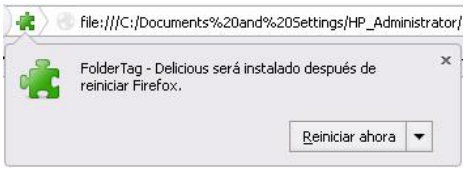

Figura A.5: Ventana para reiniciar el navegador

[1] Nuestra extensión no tiene verificado el autor porque sería necesario un certificado.

Tras el reinicio del navegador, podremos comprobar que la extensión se ha instalado correctamente en el administrador de complementos, por lo que ya estará lista para su utilización. Lo podremos comprobar en **Firefox** > **Complementos**. Ver figura A.6.

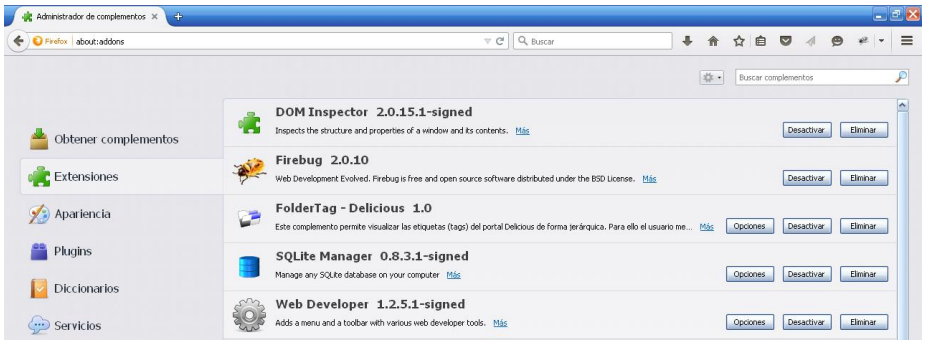

Figura A.6: Administrador de complementos

A.2. Manual de usuario

Una vez instalada la extensión *FolderTag - Delicious*, ya podremos empezar a utilizarla para personalizar el sistema de clasificación en el portal *Delicious*. A las funcionalidades de la extensión se accede principalmente a través del botón secundario del ratón una vez que se está dentro del portal *Delicious*, ya que en otras páginas el botón está deshabilitado. Ver figura A.7.

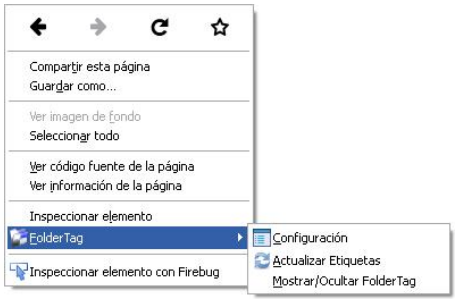

Figura A.7: Funcionalidades de la extensión

Lo primero que tendremos que hacer para empezar a utilizarla es guardar nuestro identificador del portal *Delicious* en el panel de configuración de la extensión. Seleccionaremos la opción **FolderTag** > **Configuración**. Ver figura A.8.

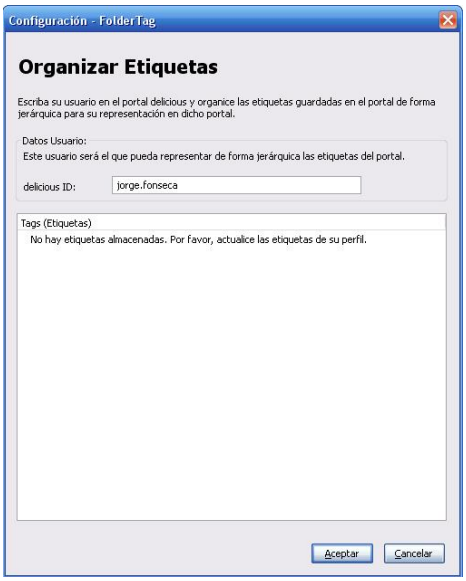

Figura A.8: Almacenar identificador en la ventana de configuración

Una vez guardado nuestro identificador podremos actualizar las etiquetas de nuestro perfil para almacenarlas y poder organizarlas jerárquicamente posteriormente. Para ello, pulsaremos con el botón secundario del ratón dentro del portal *Delicious* y seleccionaremos la acción **FolderTag** > **Actualizar Etiquetas**. Y se nos abrirá un aviso para confirmar la acción. Ver figura A.9.

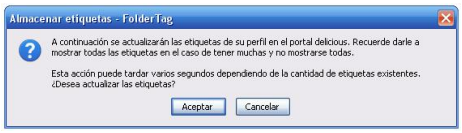

Figura A.9: Aviso almacenar las etiquetas

Cuando hayamos confirmado la acción pulsando en **Aceptar**, automáticamente se almacenarán todas las etiquetas de nuestro perfil. Es posible que cuando vayamos a actualizar las etiquetas nos encontremos con algún aviso de error en casos como:

- No haber definido ningún identificador de *Delicious*.
- No estar logueado en el portal *Delicious*.
- Almacenar incorrectamente nuestro identificador de *Delicious*.
- No encontrarse en la página donde se encuentren todas nuestras etiquetas.

Para ello se nos mostrarán avisos indicando cuál es el error. Ver figura A.10.

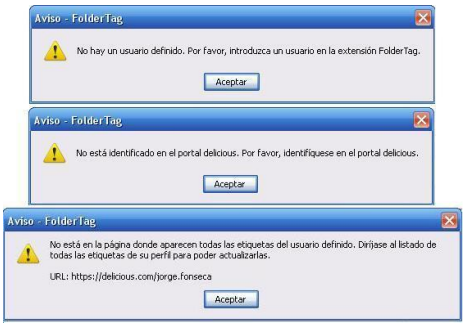

Figura A.10: Avisos de error

Una vez almacenadas todas las etiquetas de nuestro perfil, ya podremos organizarlas jerárquicamente en la ventana de configuración. Seleccionaremos la opción **FolderTag** > **Configuración** y mediante la funcionalidad de "arrastrar y soltar" iremos relacionando todas las etiquetas de forma jerárquica. Para ayudar en la ordenación, en la esquina inferior derecha de la ventana se nos indicará donde se soltará la etiqueta seleccionada. Al finalizar la ordenación de las etiquetas pulsaremos en el botón **Aceptar** para guardar los cambios realizados. Ver figura A.11.

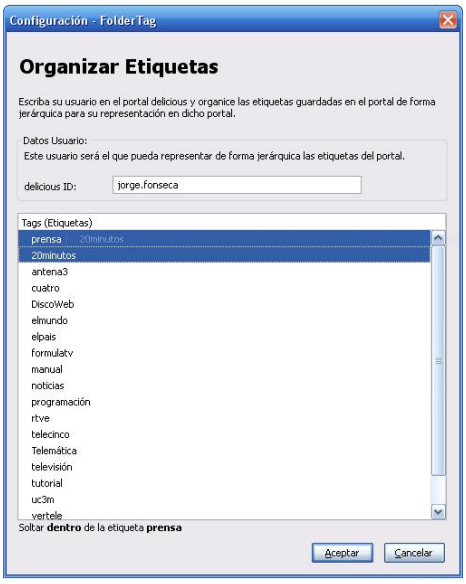

Figura A.11: Gestionar etiquetas en la ventana de configuración

Finalizada la organización de las etiquetas ya sólo faltaría representarlas en

el portal *Delicious*. Para ello activaremos la opción de mostrar el nuevo listado jerárquico de las etiquetas seleccionando **FolderTag** > **Mostrar/Ocultar FolderTag**. Una vez activado dicha opción, nos aparecerá el nuevo listado personalizado. Ver figura A.12.

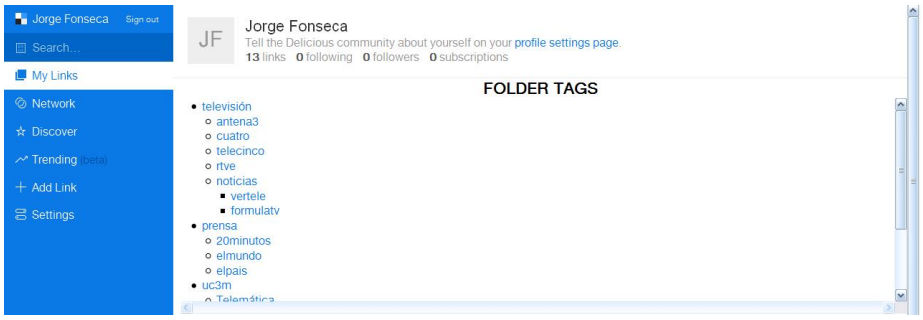

Figura A.12: Visualización de las etiquetas en forma jerárquica